

FOCUS for S/390

Creating Reports
Version 7.2

Cactus, EDA/SQL, FIDEL, FOCCALC, FOCUS, FOCUS Fusion, FOCUS Vision, Hospital-Trac, Information Builders, the Information Builders logo, Parlay, PC/FOCUS, SmartMart, SmartMode, SNAPpack, TableTalk, WALDO, Web390, WebFOCUS and WorldMART are registered trademarks and EDA, iWay, and iWay Software are trademarks of Information Builders, Inc.

Acrobat and Adobe are registered trademarks of Adobe Systems Incorporated.

Allaire and JRun are trademarks of Allaire Corporation.

NOMAD is a registered trademark of Aonix.

UniVerse is a registered trademark of Ardent Software, Inc.

IRMA is a trademark of Attachmate Corporation.

Baan is a registered trademark of Baan Company N.V.

SUPRA and TOTAL are registered trademarks of Cincom Systems, Inc.

Impromptu is a registered trademark of Cognos.

Alpha, DEC, DECnet, NonStop, and VAX are registered trademarks and Tru64, OpenVMS, and VMS are trademarks of Compaq Computer Corporation.

CA-ACF2, CA-Datcom, CA-IDMS, CA-Top Secret, & Ingres are registered trademarks of Computer Associates International, Inc.

MODEL 204 and M204 are registered trademarks of Computer Corporation of America.

Paradox is a registered trademark of Corel Corporation.

StorHouse is a registered trademark of FileTek, Inc.

HP MPE/iX is a registered trademark of Hewlett Packard Corporation.

Informix is a registered trademark of Informix Software, Inc.

ACF/VTAM, AIX, AS/400, CICS, DB2, DRDA, Distributed Relational Database Architecture, IBM, MQSeries, MVS/ESA, OS/2, OS/390, OS/400, RACF, RS/6000, S/390, VM/ESA, VSE/ESA and VTAM are registered trademarks and DB2/2, Hiperspace, IMS, MVS, QMF, SQL/DS, WebSphere, z/OS and z/VM are trademarks of International Business Machines Corporation.

INTERSOLVE and Q+E are registered trademarks of INTERSOLVE.

Orbit is a registered trademark of Iona Technologies Inc.

Approach and DataLens are registered trademarks of Lotus Development Corporation.

ObjectView is a trademark of Matesys Corporation.

ActiveX, FrontPage, Microsoft, MS-DOS, PowerPoint, Visual Basic, Visual C++, Visual FoxPro, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

Teradata is a registered trademark of NCR International, Inc.

Netscape, Netscape FastTrack Server, and Netscape Navigator are registered trademarks of Netscape Communications Corporation.

CORBA is a trademark of Object Management Group, Inc.

Oracle is a registered trademark and Rdb is a trademark of Oracle Corporation.

PeopleSoft is a registered trademark of PeopleSoft, Inc.

INFOAccess is a trademark of Pioneer Systems, Inc.

Progress is a registered trademark of Progress Software Corporation.

Red Brick Warehouse is a trademark of Red Brick Systems.

SAP and SAP R/3 are registered trademarks and SAP Business Information Warehouse and SAP BW are trademarks of SAP AG.

Silverstream is a trademark of Silverstream Software.

ADABAS is a registered trademark of Software A.G.

CONNECT:Direct is a trademark of Sterling Commerce.

Java and all Java-based marks, NetDynamics, Solaris, SunOS, and iPlanet are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

PowerBuilder and Sybase are registered trademarks and SQL Server is a trademark of Sybase, Inc.

Unicode is a trademark of Unicode, Inc.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Due to the nature of this material, this document refers to numerous hardware and software products by their trade names. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies. It is not this publisher's intent to use any of these names generically. The reader is therefore cautioned to investigate all claimed trademark rights before using any of these names other than to refer to the product described.

Copyright © 2001 by Information Builders, Inc. All rights reserved. This manual, or parts thereof, may not be reproduced in any form without the written permission of Information Builders, Inc.

Printed in the U.S.A.

Preface

This documentation describes FOCUS Reporting environments and features for FOCUS® Version 7.2. It is intended for any FOCUS user who will access corporate data to produce reports. This manual is part of the FOCUS for S/390 documentation set.

References to MVS apply to all supported versions of the OS/390, z/OS, and MVS operating environments. References to VM apply to all supported versions of the VM/ESA and z/VM operating environments.

The documentation set consists of the following components:

- The *Creating Reports* manual describes FOCUS Reporting environments and features.
- The *Describing Data* manual explains how to create the metadata for the data sources that your FOCUS procedures will access.
- The *Developing Applications* manual describes FOCUS Application Development tools and environments.
- The *Maintaining Databases* manual describes FOCUS data management facilities and environments.
- The *Using Functions* manual describes internal functions and user-written subroutines.
- The *Overview and Operating Environments* manual contains an introduction to FOCUS and FOCUS tools and describes how to use FOCUS in the VM/CMS and MVS (OS/390) environments.

The users' documentation for FOCUS Version 7.2 is organized to provide you with a useful, comprehensive guide to FOCUS.

Chapters need not be read in the order in which they appear. Though FOCUS facilities and concepts are related, each chapter fully covers its respective topic. To enhance your understanding of a given topic, references to related topics throughout the documentation set are provided. The following pages detail documentation organization and conventions.

How This Manual Is Organized

This manual is organized as follows:

Chapter/Appendix		Contents
1	<i>Creating Tabular Reports</i>	Provides an introduction to the TABLE command, a powerful tool for analyzing data.
2	<i>Displaying Report Data</i>	Describes ways to retrieve field values from a database and display them.
3	<i>Viewing and Printing Report Output</i>	Describes the HotScreen facility for viewing report output.
4	<i>Sorting Tabular Reports</i>	Describes how to display report information grouped in a particular order by sorting.
5	<i>Selecting Records for Your Report</i>	Describes how to use and specify selection criteria to display only the field values that meet your needs.
6	<i>Creating Temporary Fields</i>	Describes how to use the DEFINE and COMPUTE commands to create temporary fields.
7	<i>Including Totals and Subtotals</i>	Describes how to use subtotals and grand totals to summarize numeric information and aid in interpreting detailed information in a report.
8	<i>Using Expressions</i>	Describes how to combine operators, fieldnames, and constants in an expression to derive new values.
9	<i>Customizing Tabular Reports</i>	Describes how to override the default report formats to meet your individual presentation needs.
10	<i>Styling Reports: StyleSheets</i>	Describes how to visually style your reports with StyleSheets, used to control report output to be printed on a PostScript printer.

Chapter/Appendix		Contents
11	<i>Saving and Reusing Report Output</i>	Describes how to save report output in a wide variety of formats.
12	<i>Handling Records With Missing Field Values</i>	Describes how missing data affects report results and how to treat and represent it.
13	<i>Joining Data Sources</i>	Describes how to join two or more related data sources to create a larger integrated data structure from which you can report.
14	<i>Merging Data Sources</i>	Describes how to merge and concatenate two or more data sources into a new permanent data file.
15	<i>Improving Report Processing</i>	Describes methods of increasing data retrieval and reporting efficiency.
16	<i>Creating Financial Reports</i>	Describes Financial Modeling Language (FML), used to create and present financially oriented data, using inter-row calculations.
17	<i>Creating a Free-Form Report</i>	Describes how to present data in an unrestricted (non-tabular) format.
18	<i>Creating Graphs: GRAPH</i>	Describes the FOCUS GRAPH facility, which you can use to display data in graph format instead of tabular format.
19	<i>Using SQL to Create Reports</i>	Describes how to use SQL to retrieve and analyze FOCUS and RDBMS data.
A	<i>Master Files and Diagrams</i>	Contains Master Files and diagrams of sample databases used in the documentation examples.
B	<i>Error Messages</i>	Describes how to access FOCUS error messages.
C	<i>Syntax Summary</i>	Summarizes FOCUS Table commands and options.
D	<i>Writing User-Coded Programs to Create HOLD Files</i>	Describes how to write programs that get records retrieved by FOCUS so you can write them to files in a custom format.
E	<i>Character Charts</i>	Lists EBCDIC codes and their corresponding character representations.

Summary of New Features

The new FOCUS features and enhancements described in this documentation set are listed in the following table.

New Feature	Manual	Chapter
Field-based Reformatting	<i>Creating Reports</i>	Chapter 1, <i>Creating Tabular Reports</i>
Increased Report Width	<i>Creating Reports</i>	Chapter 1, <i>Creating Tabular Reports</i>
ACROSS-TOTAL	<i>Creating Reports</i>	Chapter 4, <i>Sorting Tabular Reports</i>
Tiles	<i>Creating Reports</i>	Chapter 4, <i>Sorting Tabular Reports</i>
DEFINE FILE SAVE and DEFINE FILE RETURN	<i>Creating Reports</i>	Chapter 6, <i>Creating Temporary Fields</i>
Forecast	<i>Creating Reports</i>	Chapter 6, <i>Creating Temporary Fields</i>
Creating Comma-Delimited Files	<i>Creating Reports</i>	Chapter 11, <i>Saving and Reusing Report Output</i>
Creating Tab-Delimited Files	<i>Creating Reports</i>	Chapter 11, <i>Saving and Reusing Report Output</i>
Long Master File Names	<i>Creating Reports</i>	Chapter 11, <i>Saving and Reusing Report Output</i>
JOIN WHERE	<i>Creating Reports</i>	Chapter 13, <i>Joining Data Sources</i>
KEEPDEFINES	<i>Creating Reports</i>	Chapter 13, <i>Joining Data Sources</i>
Long Master File Names	<i>Describing Data</i>	Chapter 1, <i>Understanding a Data Source Description</i>
4K Alpha Fields	<i>Describing Data</i>	Chapter 4, <i>Describing an Individual Field</i>
Extended Currency Symbol Support	<i>Describing Data</i>	Chapter 4, <i>Describing an Individual Field</i>

New Feature	Manual	Chapter
SUFFIX = COMT/COMMA/TABT	<i>Describing Data</i>	Chapter 5, <i>Describing a Sequential, VSAM, or ISAM Data Source</i>
AUTODATE	<i>Describing Data</i>	Chapter 6, <i>Describing a FOCUS Data Source</i>
CDN	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
CENT-ZERO	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
Exit on Error	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
KEEPDEFINES	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
PCOMMA	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
Unlimited -INCLUDEs	<i>Developing Applications</i>	Chapter 3, <i>Managing an Application With Dialogue Manager</i>
SQUEEZ Function	<i>Using Functions</i>	Chapter 3, <i>Character Functions</i>
STRIP Function	<i>Using Functions</i>	Chapter 3, <i>Character Functions</i>
TRIM Function	<i>Using Functions</i>	Chapter 3, <i>Character Functions</i>
DYNAM ALLOC LONGNAME	<i>Overview and Operating Environments</i>	Chapter 5, <i>OS/390 and MVS Guide to Operations</i>

Documentation Conventions

The following conventions apply throughout this manual:

Convention	Description
<code>THIS TYPEFACE or this typeface</code>	Denotes syntax that you must enter exactly as shown.
<code>this typeface</code>	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
<code><u>underscore</u></code>	Indicates a default setting.
<code>this typeface</code>	Represents a placeholder (or variable) in a text paragraph, indicates a cross-reference, or emphasizes an important term.
this typeface	Highlights file names and commands (in a text paragraph) that must be lowercase.
this typeface	Indicates buttons, menu items, and dialog box options you can click or select.
Key + Key	Indicates keys that must be pressed simultaneously.
{ }	Indicates two choices from which you must choose one. You type one of these choices, not the braces.
[]	Indicates a group of optional parameters. None are required, but you may select one of them. Type only the information within the brackets, not the brackets.
	Separates two mutually exclusive choices in a syntax line. You type one of these choices, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the parameters, not the ellipsis points (...).
· · ·	Indicates that there are (or could be) intervening or additional commands.

Related Publications

Visit our World Wide Web site, <http://www.informationbuilders.com>, to view a current listing of our publications and to place an order. To obtain a print catalog, contact the Publications Order Department at (800) 969-4636.

Customer Support

Do you have questions about FOCUS?

Call Information Builders Customer Support Service (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your FOCUS questions. Information Builders consultants can also give you general guidance regarding product capabilities and documentation. Please be ready to provide your six-digit site code number (xxxx.xx) when you call.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our World Wide Web site, <http://www.informationbuilders.com>. It connects you to the tracking system and known-problem database at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of www.informationbuilders.com also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse Online, or call (800) 969-INFO.

Information You Should Have

To help our consultants answer your questions most effectively, be ready to provide the following information when you call:

- Your six-digit site code number (xxxx.xx).
- The FOCEXEC procedure (preferably with line numbers).
- Master file with picture (provided by CHECK FILE).

- Run sheet (beginning at login, including call to FOCUS), containing the following information:
 - ? RELEASE
 - ? FDT
 - ? LET
 - ? LOAD
 - ? COMBINE
 - ? JOIN
 - ? DEFINE
 - ? STAT
 - ? SET/? SET GRAPH
 - ? USE
 - ? TSO DDNAME OR CMS FILEDEF
- The exact nature of the problem:
 - Are the results or the format incorrect? Are the text or calculations missing or misplaced?
 - The error message and code, if applicable.
 - Is this related to any other problem?
- Has the procedure or query ever worked in its present form? Has it been changed recently? How often does the problem occur?
- What release of the operating system are you using? Has it, FOCUS, your security system, or an interface system changed?
- Is this problem reproducible? If so, how?
- Have you tried to reproduce your problem in the simplest form possible? For example, if you are having problems joining two data sources, have you tried executing a query containing just the code to access the data source?
- Do you have a trace file?
- How is the problem affecting your business? Is it halting development or production? Do you just have questions about functionality or documentation?

User Feedback

The Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual. Please use the Document Feedback form on our Web site, <http://www.informationbuilders.com/bookstore/derf.html>. You can also use the Reader Comments form at the end of this manual to relay suggestions for improving the publication or to alert us to corrections.

Thank you, in advance, for your comments.

Information Builders Consulting and Training

Interested in training? Information Builders Education Department offers a wide variety of training courses for this and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our World Wide Web site (<http://www.informationbuilders.com>) or call (800) 969-INFO to speak to an Education Representative.

Contents

1	Creating Tabular Reports	1-1
	Requirements for Creating a Report	1-2
	Creating a Report Request.....	1-3
	Beginning a Report Request.....	1-3
	Requesting Help When Issuing a Report Request.....	1-4
	Completing a Report Request.....	1-5
	Selecting a Report Output Destination	1-5
	The Parts of a Report Request	1-6
	Displaying Data.....	1-6
	Sorting a Report	1-7
	Selecting Records.....	1-10
	Showing Subtotals and Totals	1-10
	Creating Temporary Fields.....	1-12
	Including Display Fields in a Report Request	1-13
	Referring to Fields in a Report Request	1-13
	Referring to an Individual Field	1-14
	Referring to Fields Using Long and Qualified Field Names.....	1-14
	Referring to All of the Fields in a Segment.....	1-16
	Displaying a List of Field Names	1-16
	Listing Field Names, Aliases, and Format Information	1-17
	Customizing a Report.....	1-17
	Changing the Format of a Report Column	1-19
	Field-Based Reformatting	1-21
	Determining the Width of a Report Column	1-24
	Saving and Reusing Report Output	1-24
2	Displaying Report Data.....	2-1
	Displaying Individual Values	2-3
	Displaying All Fields	2-5
	Displaying the Structure of a Multi-Path Data Source	2-6
	Adding Values.....	2-9
	Counting Values	2-10
	Counting Segment Instances	2-12
	Expanding Byte Precision for COUNT and LIST	2-13

Manipulating Display Fields With Prefix Operators	2-15
Averaging Values of a Field.....	2-17
Averaging the Sum of Squared Fields.....	2-17
Calculating Maximum and Minimum Field Values	2-18
Calculating Column and Row Percents	2-19
Producing a Direct Percent of a Count.....	2-21
Aggregating and Listing Unique Values	2-21
Retrieving First and Last Records	2-24
Summing and Counting Values.....	2-27
Manipulating Display Field Values in a Sort Group	2-29
3 Viewing and Printing Report Output	3-1
Displaying Reports in Hot Screen	3-2
Using PRINTPLUS	3-4
Controlling the Display of Empty Reports	3-6
Accessing Help Information.....	3-6
Scrolling a Report.....	3-7
Scrolling Forward.....	3-7
Scrolling Backward.....	3-7
Scrolling Horizontally	3-8
Scrolling From Fixed Columns (Fencing).....	3-8
Scrolling Report Headings	3-9
Saving Selected Data.....	3-9
Locating Character Strings.....	3-10
Repeating Commands.....	3-10
Redisplaying Reports	3-11
Previewing Your Report.....	3-12
Displaying BY Fields With Panels.....	3-12
Scrolling by Columns of BY Fields in Panels	3-14
The SET COLUMNS Command.....	3-14
Displaying Reports in the Panel Facility	3-15
Printing Reports.....	3-16
The OFFLINE Command.....	3-16
Printing Reports in Hot Screen.....	3-17
Displaying Reports in the Terminal Operator Environment.....	3-17
4 Sorting Tabular Reports.....	4-1
Sorting Rows	4-3
Using Multiple Sort Fields With BY	4-4
Sorting Columns.....	4-5
Using Multiple Sort Fields With ACROSS.....	4-7
Producing Column Totals With ACROSS-TOTAL	4-7
Sorting Rows and Columns.....	4-9

Specifying the Sort Order	4-10
Specifying Your Own Sort Order	4-12
Grouping Numeric Data Into Ranges	4-16
Grouping Numeric Data Into Tiles	4-19
Restricting Sort Field Values by Highest/Lowest Rank	4-24
Aggregating and Sorting Report Columns	4-25
Ranking Sort Field Values	4-27
Hiding Sort Values	4-29
Sorting With Multiple Display Commands	4-31
Improving Efficiency With External Sorts	4-33
Aggregation by External Sort	4-35
Changing Retrieval Order With Aggregation	4-37
Using External Sorts to Extract Data	4-37
Estimating SORTWORK Sizes for an External Sort	4-38
Displaying External Sort Messages	4-39
5 Selecting Records for Your Report	5-1
Choosing a Filtering Method	5-2
Selections Based on Individual Values	5-2
Controlling Record Selection in Multi-Path Data Sources	5-5
Selection Based on Aggregate Values	5-10
Using Compound Expressions for Record Selection	5-12
Using Operators in Record Selection Tests	5-13
Types of Record Selection Tests	5-16
Range Tests With FROM and TO	5-16
Range Tests With GE and LE or GT and LT	5-17
Missing Data Tests	5-19
Character String Screening With CONTAINS and OMITS	5-19
Screening on Masked Fields With LIKE and IS	5-20
Using an Escape Character for LIKE	5-24
Qualifying Parent Segments Using INCLUDES and EXCLUDES	5-25
Selections Based on Group Key Values	5-26
Setting Limits on the Number of Records Read	5-27
Selecting Records Using IF Phrases	5-28
Reading Selection Values From a File	5-29
Assigning Screening Conditions to a File	5-32
Applying Filters to Joined Structures	5-38
VSAM Record Selection Efficiencies	5-39
Reporting From Files With Alternate Indexes	5-39

6	Creating Temporary Fields	6-1
	The Difference Between DEFINE and COMPUTE	6-2
	Defining a Virtual Field.....	6-4
	Defining Multiple Virtual Fields	6-7
	Establishing a Segment Location for a Virtual Field.....	6-8
	Defining Virtual Fields Using a Multi-Path Data Source.....	6-9
	Increasing the Speed of DEFINE Calculations	6-10
	Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN	6-10
	Computing Calculated Values	6-11
	Using Positional Column Referencing With Calculated Values	6-14
	Using COMPUTE and ACROSS	6-15
	Sorting Calculated Values	6-15
	Screening on Calculated Values.....	6-16
	Calculating Trend Values and Forecasts	6-17
	FORECAST Processing	6-17
	Forecasting Methods	6-21
	Using a Simple Moving Average	6-21
	Using an Exponential Moving Average	6-25
	Using a Linear Regression Equation	6-27
	FORECAST Reporting Techniques	6-29
	Using Functions With Temporary Fields	6-32
	Creating Temporary Fields Unrelated to Master Files	6-33
7	Including Totals and Subtotals.....	7-1
	Calculating Row and Column Totals.....	7-2
	Adding Section Totals and a Grand Total	7-6
	Including Subtotals.....	7-8
	Recalculating Values for Subtotal Rows	7-12
	Performing Calculations at Sort Field Breaks	7-16
	Suppressing Grand Totals.....	7-19
	Conditionally Displaying Summary Lines and Text	7-21
8	Using Expressions.....	8-1
	Using Expressions in Commands and Phrases	8-2
	Types of Expressions.....	8-3
	Expressions and Field Formats.....	8-3
	Creating a Numeric Expression.....	8-4
	Order of Evaluation.....	8-5

Creating a Date or Date-Time Expression.....	8-7
Formats for Date Values.....	8-8
Performing Calculations on Dates.....	8-9
Cross-Century Dates With DEFINE and COMPUTE.....	8-10
Returned Field Format Selection.....	8-10
Using a Date Constant in an Expression	8-10
Extracting a Date Component	8-11
Combining Fields With Different Formats in an Expression	8-11
Creating a Character Expression	8-12
Embedding a Quotation Mark in a Quote-Delimited Literal String	8-12
Concatenating Character Strings	8-13
Creating a Logical Expression.....	8-14
Creating a Conditional Expression	8-16
9 Customizing Tabular Reports.....	9-1
Creating Paging and Numbering	9-2
Specifying a Page Break: PAGE-BREAK	9-2
Inserting Page Numbers: TABPAGENO	9-4
Suppressing Page Numbers: SET PAGE.....	9-5
Preventing an Undesirable Split.....	9-5
Separating Sections of a Report: SKIP-LINE and UNDER-LINE.....	9-8
Adding Blank Lines: SKIP-LINE	9-8
Underlining Values: UNDER-LINE	9-10
Suppressing Fields: SUP-PRINT or NOPRINT	9-11
Creating New Column Titles: AS.....	9-15
Customizing Column Names: SET QUALTITLES	9-17
Positioning Columns: IN	9-18
Reducing a Report's Width: FOLD-LINE and OVER.....	9-22
Compressing the Columns of Reports: FOLD-LINE	9-22
Decreasing the Width of a Report: OVER	9-23
Controlling Column Spacing: SET SPACES	9-25
Column Title Justification	9-26
Customizing Reports With SET Parameters.....	9-27
Producing Headings and Footings.....	9-29
Report and Page Headings	9-30
Report and Page Footings.....	9-33
Subheads	9-35
Subfoots.....	9-36
Positioning Text	9-39
Using Data in Headings and Footings	9-41
Producing a Free-Form Report.....	9-45

Conditionally Formatting Reports With the WHEN Clause.....	9-46
Controlling the Display of Empty Reports	9-52
10 Styling Reports: StyleSheets	10-1
Introduction to StyleSheets.....	10-2
What Is a StyleSheet?.....	10-5
What Is a Style?.....	10-5
When You Need to Create a StyleSheet File.....	10-6
Comparison of Reports With and Without StyleSheets	10-7
Creating a StyleSheet	10-9
Creating a StyleSheet Within a Report Request.....	10-9
Activating an Existing StyleSheet File.....	10-11
Printing Styled Reports.....	10-12
Styling the Page Layout.....	10-13
Displaying Current Settings: The ? STYLE Query	10-17
StyleSheet Files	10-17
StyleSheet Syntax.....	10-18
Checking StyleSheet Syntax	10-19
Style Definitions.....	10-19
Identifying Report Components	10-21
Selecting and Manipulating Report Components.....	10-24
Selecting Headings and Footings	10-28
Selecting Report Columns.....	10-34
Positioning Headings, Footings, and Columns.....	10-38
Determining Column Widths	10-39
Changing Column Sequence	10-40
Specifying Column Spacing.....	10-42
StyleSheet Inheritance	10-43
Conditional Styling.....	10-49
11 Saving and Reusing Report Output	11-1
Saving Your Report Output.....	11-2
Creating HOLD and PCHOLD Files.....	11-3
Holding Report Output in FOCUS Format.....	11-9
Controlling Attributes in HOLD Master Files.....	11-14
Controlling Field Names in a HOLD Master File	11-14
Controlling Fields in a HOLD Master File.....	11-18
Controlling the TITLE and ACCEPT Attributes in the HOLD Master File.....	11-19
Keyed Retrieval From HOLD Files.....	11-21
Creating SAVE and SAVB Files.....	11-23
Choosing Output File Formats	11-26

Saving Report Output in INTERNAL Format.....	11-42
12 Handling Records With Missing Field Values	12-1
Irrelevant Report Data	12-2
Missing Field Values	12-3
MISSING Attribute in the Master File.....	12-4
MISSING Attribute in a DEFINE Command.....	12-5
Testing for a Segment With a Missing Field Value	12-9
Preserving Missing Data Values in an Output File	12-12
Handling a Missing Segment Instance	12-13
Including Missing Instances in Reports With the ALL. Prefix	12-16
Including Missing Instances in Reports With the SET ALL Command	12-17
Testing for Missing Instances in FOCUS Data Sources.....	12-19
Setting the NODATA Character String	12-20
13 Joining Data Sources.....	13-1
Types of Joins.....	13-2
Unique and Non-Unique Joined Structures	13-4
Recursive Joined Structures	13-7
How the JOIN Command Works.....	13-12
Creating an Equijoin.....	13-13
Joining From a Virtual Field to a Real Field Using an Equijoin	13-17
Data Formats of Shared Fields	13-21
Joining Fields With Different Numeric Data Types.....	13-22
Using a Conditional Join	13-23
Preserving Virtual Fields During Join Parsing	13-26
Preserving Virtual Fields Using KEEPDEFINES	13-26
Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN	13-29
Screening Segments With Conditional JOIN Expressions	13-31
Parsing WHERE Criteria in a Join	13-31
Displaying Joined Structures	13-31
Clearing Joined Structures.....	13-34
Clearing a Conditional Join	13-34
14 Merging Data Sources.....	14-1
Merging Data.....	14-2
MATCH Processing	14-5
MATCH Processing With Common High Order Sort Fields	14-9
Fine Tuning MATCH Processing.....	14-12
Universal Concatenation	14-14
Field Name and Format Matching.....	14-17

Merging Concatenated Data Sources	14-19
Using Sort Fields in MATCH Requests	14-22
Cartesian Product	14-25
15 Improving Report Processing.....	15-1
Rotating a Data Structure for Enhanced Retrieval.....	15-2
Optimizing Retrieval Speed for FOCUS Data Sources	15-4
Automatic Indexed Retrieval.....	15-4
Data Retrieval Using TABLEF	15-6
Preserving the Internal Matrix of Your Last Report.....	15-7
16 Creating Financial Reports.....	16-1
Reporting With FML.....	16-2
Creating Rows From Data	16-5
Changing Row Titles.....	16-7
Creating Rows From Multiple Records.....	16-8
Using the BY Phrase in FML Requests.....	16-10
Performing Inter-Row Calculations.....	16-12
Referring to Rows	16-13
Referring to Columns	16-16
Referring to Column Numbers	16-16
Referring to Contiguous Columns.....	16-17
Referring to Column Addresses	16-18
Referring to Relative Column Addresses	16-19
Referring to Column Values.....	16-20
Referring to Cells	16-20
Using Subroutines in Calculations	16-22
Supplying Data Directly in the FML Request	16-24
Inserting Rows of Free Text	16-25
Adding Columns to an FML Report.....	16-27
Creating Recursive Models	16-28
Formatting an FML Report	16-30
Suppressing Tagged Rows	16-31
Suppressing Rows With No Data	16-32
Saving and Retrieving Intermediate Report Results.....	16-32
Posting Data	16-33
Creating HOLD Files From FML Reports	16-35

17	Creating a Free-Form Report	17-1
	Introduction to Free-Form Reports.....	17-2
	Designing a Free-Form Report.....	17-6
	Incorporating Text in a Free-Form Report	17-6
	Incorporating Data Fields in a Free-Form Report	17-7
	Incorporating Graphic Characters in a Free-Form Report.....	17-7
	Laying Out a Free-Form Report.....	17-8
	Sorting and Selecting Records in a Free-Form Report.....	17-8
18	Creating Graphs: GRAPH	18-1
	Introduction	18-2
	GRAPH vs. TABLE Requests.....	18-2
	Controlling the Format.....	18-7
	Graphic Devices Supported.....	18-10
	Command Syntax	18-12
	GRAPH vs. TABLE Syntax.....	18-12
	Specifying Graph Forms and Contents.....	18-14
	Graph Forms.....	18-21
	Connected Point Plots	18-22
	Histograms	18-26
	Bar Charts.....	18-29
	Pie Charts	18-33
	Scatter Diagrams	18-35
	Adjusting Graph Elements	18-38
	The Horizontal Axis: System Defaults.....	18-40
	The Vertical Axis: System Defaults	18-43
	Highlighting Facilities.....	18-45
	Special Topics	18-46
	Plotting Dates	18-47
	Handling Missing Data.....	18-48
	Using Fixed-Axis Scales	18-50
	Saving Formatted GRAPH Output.....	18-51
	Displaying Graphs With PC/FOCUS or FOCUS for Windows	18-53
	Creating Formatted Input for CA-TELLAGRAF.....	18-53
	Using the FOCUS ICU Interface.....	18-54
	Special Graphics Devices	18-54
	Medium-Resolution Devices.....	18-55
	High-Resolution Devices	18-55
	Command and SET Parameter Summary	18-57
	GRAPH Command.....	18-57
	SET Parameters	18-60

19	Using SQL to Create Reports	19-1
	Supported and Unsupported SQL Statements	19-2
	Using SQL Translator Commands	19-5
	Automatic Passthru	19-6
	The SQL SELECT Statement.....	19-7
	SQL Joins	19-8
	SQL CREATE TABLE and INSERT Commands	19-11
	SQL CREATE VIEW and DROP VIEW Commands.....	19-12
	Cartesian Product Style Answer Sets	19-13
	Continental Decimal Notation (CDN).....	19-13
	Specifying Field Names in SQL Requests	19-14
	SQL UNION, INTERSECT, and EXCEPT Operators.....	19-15
	Numeric Constants, Literals, Expressions, and Functions.....	19-15
	SQL Translator Support for Date, Time, and Timestamp Fields.....	19-15
	Extracting Date-Time Components Using the SQL Translator.....	19-17
	Index Optimized Retrieval	19-20
	Optimized Joins.....	19-20
	TABLEF Optimization.....	19-21
	SQL INSERT, UPDATE, and DELETE Commands	19-22
A	Master Files and Diagrams.....	A-1
	Creating Sample Data Sources	A-2
	The EMPLOYEE Data Source	A-3
	The EMPLOYEE Master File	A-4
	The EMPLOYEE Structure Diagram.....	A-5
	The JOBFIL Data Source.....	A-6
	The JOBFIL Master File.....	A-6
	The JOBFIL Structure Diagram.....	A-6
	The EDUCFILE Data Source.....	A-7
	The EDUCFILE Master File	A-7
	The EDUCFILE Structure Diagram.....	A-7
	The SALES Data Source	A-8
	The SALES Master File	A-8
	The SALES Structure Diagram	A-9
	The PROD Data Source.....	A-10
	The PROD Master File.....	A-10
	The PROD Structure Diagram.....	A-10
	The CAR Data Source	A-11
	The CAR Master File	A-11
	The CAR Structure Diagram	A-12

The LEDGER Data Source	A-13
The LEDGER Master File.....	A-13
The LEDGER Structure Diagram	A-13
The FINANCE Data Source	A-14
The FINANCE Master File	A-14
The FINANCE Structure Diagram	A-14
The REGION Data Source	A-15
The REGION Master File	A-15
The REGION Structure Diagram	A-15
The COURSES Data Source	A-16
The COURSES Master File	A-16
The COURSES Structure Diagram	A-16
The EMPDATA Data Source	A-17
The EMPDATA Master File	A-17
The EMPDATA Structure Diagram	A-17
The EXPERSON Data Source.....	A-18
The EXPERSON Master File.....	A-18
The EXPERSON Structure Diagram.....	A-18
The TRAINING Data Source	A-19
The TRAINING Master File	A-19
The TRAINING Structure Diagram	A-19
The PAYHIST File.....	A-20
The PAYHIST Master File.....	A-20
The PAYHIST Structure Diagram	A-20
The COMASTER File	A-21
The COMASTER Master File.....	A-22
The COMASTER Structure Diagram	A-23
The VideoTrk and MOVIES Data Sources	A-24
VideoTrk Master File	A-24
MOVIES Master File	A-24
VideoTrk Structure Diagram.....	A-25
MOVIES Structure Diagram	A-26
The VIDEOTR2 Data Source.....	A-26
The VIDEOTR2 Master File	A-26
The VIDEOTR2 Access File.....	A-27
The VIDEOTR2 Structure Diagram.....	A-28
The Gotham Grinds Data Sources	A-29
The GGDEMOG Data Source.....	A-29
The GGORDER Data Source.....	A-30
The GGPRODS Data Source	A-31
The GGSALES Data Source	A-32
The GGSTORES Data Source.....	A-33

B Error Messages B-1
 Accessing Error Files B-2
 Displaying Messages Online B-3

C Syntax Summary C-1
 TABLE Syntax Summary C-2
 TABLEF Syntax Summary C-3
 MATCH Syntax Summary C-4
 FOR Syntax Summary C-5

D Writing User-Coded Programs to Create HOLD Files D-1
 Arguments Used in Calls to Programs That Create HOLD Files D-2

E Character Charts..... E-1
 Letters E-2
 Numbers E-2
 Punctuation E-3
 Symbols E-3
 Accent Marks and Accented Letters E-4

Index I-1

CHAPTER 1

Creating Tabular Reports

Topics:

- Requirements for Creating a Report
- Creating a Report Request
- The Parts of a Report Request
- Including Display Fields in a Report Request
- Referring to Fields in a Report Request
- Customizing a Report
- Changing the Format of a Report Column
- Saving and Reusing Report Output

The FOCUS reporting language is a powerful tool for analyzing and formatting information. The language is non-procedural—that is, you only need to think about what information you want to present in your report. For the most part, you can describe the report in any order—the sequence of commands is not important.

The simplest form of report that you can produce is a tabular report—that is, a report whose information is arranged vertically in columns. This is the basic report format, incorporating the fundamental reporting concepts and command syntax. Most of the other report formats build on these concepts and syntax.

Requirements for Creating a Report

To create a report, only two things are required:

- **Data.** You need data from which to report. If the data is protected by an underlying security system, you may need permission to report from the data source. In addition, the server must be able to locate the data source. See the *Developing Applications* manual for information on data source locations.

You can report from many different types of data sources (with variations for different operating environments), including the following:

- Relational data sources, such as DB2, Teradata, and Oracle, and Sybase.
- Hierarchical data sources, such as IMS and FOCUS.
- Indexed data sources, such as ISAM and VSAM.
- Network data sources, such as CA-IDMS.
- Sequential data sources, both fixed-format and comma-delimited format.
- Multi-dimensional data sources, such as Fusion.

For a complete list, see your *Describing Data* manual.

- **A data description.** You need a Master File, which describes the data source from which you are reporting. The Master File is a map of the segments in the data source and all of the fields in each segment. For some types of data sources, the Master File is supplemented by an Access File. See the *Describing Data* manual for information on Master Files and Access Files.

By looking at the Master File, you can determine what fields are in the data source, what they are named, and how they are formatted. You can also determine how the segments in the data source relate to each other. Although you can create a very simple report without this information, knowing the structure of the data source enables you to generate creative and sophisticated reports.

You can supplement the information in the Master File by generating a picture of the data source structure—that is, of how the data source segments relate to each other. Use the following command:

```
CHECK FILE filename PICTURE RETRIEVE
```

In the picture, segments are shown in the order in which they are retrieved. Four fields of each segment are displayed. For details, see Chapter 2, *Displaying Report Data*.

Creating a Report Request

You can use any text editor to create your report request. Using the text editor, you can create ad hoc reports or create a report and save it as a stored procedure, enabling you to edit the request at any time. Stored procedures are described in more detail in the *Developing Applications* manual.

Beginning a Report Request

A report request begins with the `TABLE FILE` command and ends with the `END` command. The commands and phrases between the beginning and end of a request define the contents and format of a report. These parts of the request are optional; you only need to include the commands and phrases that produce the report functions you want. For example, you need only include a sorting phrase if you want your report to be sorted.

Syntax

How to Begin a Report Request

To begin a report request, use the command

```
TABLE FILE filename
```

where:

```
filename
```

Specifies a data source for the report.

Example

Issuing Report Requests

The following examples produce the same report:

1.

```
TABLE FILE EMPLOYEE PRINT LAST_NAME BY DEPARTMENT
END
```
2.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
BY DEPARTMENT
END
```
3.

```
TABLE
FILE EMPLOYEE
PRINT
LAST_NAME BY DEPARTMENT
END
```

The output is:

DEPARTMENT	LAST_NAME
MIS	SMITH JONES MCCOY BLACKWOOD GREENSPAN CROSS
PRODUCTION	STEVENS SMITH BANNING IRVING ROMANS MCKNIGHT

Example

Reporting With a Default Data Source

An alternate way to specify the file name is with the SET FILE command. SET FILE establishes a default data source for all requests, as described in the *Developing Applications* manual. The following sets the EMPLOYEE data source as the default:

```
SET FILE = EMPLOYEE
TABLE
PRINT CURR_SAL
BY DEPARTMENT
END
```

This alternative is useful when you wish to enter several report requests against the same data source. Of course, you can still issue requests against other data sources simply by specifying the file name in the request instead of relying upon the default name.

Requesting Help When Issuing a Report Request

If you issue report requests interactively at the command prompt, rather than from a procedure, online error correction is provided with help text. For example, if you enter

```
TABLE FI EMPLOYEE
```

at the command prompt, the following error message displays:

```
(FOC001) THE NAME OF THE FILE OR THE WORD 'FILE' IS MISSING
```

Enter your correction at the REPLY prompt. For this example, the correct reply is:

```
FILE
```

However, if the information provided by the error message is not sufficient, issue

```
HELP
```

or

```
?
```

at the REPLY prompt for a more detailed explanation of the error.

Every process and command you enter is scanned, and a report is generated immediately after you enter the END or RUN command.

When the value of the MESSAGE parameter is ON (the default value), the number of records retrieved from the data source and the number of lines displayed in the report displays at the beginning of each report.

Completing a Report Request

To complete a report request, use the END or RUN command. These commands must be typed on a line by themselves. To discontinue a report request without executing it, enter the QUIT command.

If you plan to issue consecutive report requests against different data sources during one session, use the END command.

You also have the option of using the RUN command to complete a report request. The RUN command keeps the TABLE facility and the data source active for the duration of the TABLE session. This is useful since you do not need to repeat the TABLE command to produce another report using the same data source.

Selecting a Report Output Destination

Once you generate a report, you still need to display it. The following facilities are available for displaying your report:

- **On a screen.** The Hot Screen facility enables you to search for report data, save parts of the report to a file, and customize how your report scrolls on the screen. Unless you specify otherwise, your reports automatically displays on the screen using the Hot Screen facility.
- **On paper.** When you print your reports on paper, you can control how reports that are too wide to fit on a single page are arranged on the supplementary pages—repeating essential columns on each page—so that the context of the data on each page is clear.

Of course, you can easily direct the same report to both the screen and the printer by switching display modes and using the RETYPE command. Or you can choose not to display your report at all, and instead store the results as a data source using the HOLD, PCHOLD, SAVE, or SAVB command. For details see Chapter 11, *Saving and Reusing Report Output*.

The Parts of a Report Request

The commands you place in between the TABLE FILE command and the END command of your report determine the contents of your report and how it will be displayed. For example, you would enter sort commands to specify how and which fields you want your data sorted by. You can define the contents of your report by:

- **Displaying data.** You can display data in your report by listing, summing, or counting it. You can also perform more complex operations on it, such as finding the highest value of a field and calculating the average sum of squares of all the values of a field.
- **Sorting a report.** When you display information, you can sort it in almost any order that you wish.
- **Selecting records.** You can specify which records will be selected for your report.
- **Showing subtotals and totals.** You can display subtotals and totals for the columns in your report.
- **Creating temporary fields.** You can create temporary fields, deriving their values from real fields, and include them in your report.

Displaying Data

Reporting, at the simplest level, retrieves field values from a data source and displays these values. You can present these values in a number of ways:

- List each field value using the PRINT and LIST commands.
- Add all the values and display the sum using the SUM command (or its synonyms ADD or WRITE).
- Count all the values and display the quantity using the COUNT command.

For more information, see Chapter 2, *Displaying Report Data*.

Example

Displaying Data

To report on salary information for each employee, and to include each employee's ID, last name, and current salary in the report, you could use the PRINT command to print the EMP_ID, LAST_NAME, and CURR_SAL fields.

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND LAST_NAME AND CURR_SAL
END
```

Notice that each field named in the PRINT command has its own report column. The column headings and formats come from the Master File.

The output is:

PAGE 1

EMP_ID	LAST_NAME	CURR_SAL
-----	-----	-----
071382660	STEVENS	\$11,000.00
112847612	SMITH	\$13,200.00
117593129	JONES	\$18,480.00
119265415	SMITH	\$9,500.00
119329144	BANNING	\$29,700.00
123764317	IRVING	\$26,862.00
126724188	ROMANS	\$21,120.00
219984371	MCCOY	\$18,480.00
326179357	BLACKWOOD	\$21,780.00
451123478	MCKNIGHT	\$16,100.00
543729165	GREENSPAN	\$9,000.00
818692173	CROSS	\$27,062.00

Sorting a Report

Sorting a report enables you to organize a column's information in a chosen order. The sort field (the field that controls the sorting order) is displayed at the left of the report. Sort fields are displayed when their values change.

You use the BY phrase to sort information vertically down a column. Sometimes it is more effective to display information horizontally, across a row. You can do this using the ACROSS phrase. You can also combine the BY and ACROSS phrases to sort a report down columns and across rows, creating a simple matrix.

For more information, see Chapter 4, *Sorting Tabular Reports*.

Example

Sorting Report Columns

If you want to sort columns by employee name, you could issue the following request:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND CURR_SAL
BY LAST_NAME
END
```

The output is:

PAGE 1

LAST_NAME	EMP_ID	CURR_SAL
-----	-----	-----
BANNING	119329144	\$29,700.00
BLACKWOOD	326179357	\$21,780.00
CROSS	818692173	\$27,062.00
GREENSPAN	543729165	\$9,000.00
IRVING	123764317	\$26,862.00
JONES	117593129	\$18,480.00
MCCOY	219984371	\$18,480.00
MCKNIGHT	451123478	\$16,100.00
ROMANS	126724188	\$21,120.00
SMITH	112847612	\$13,200.00
	119265415	\$9,500.00
STEVENS	071382660	\$11,000.00

The sort field (the field that controls the sorting order) is displayed at the left of the report. Note that two rows of the report are assigned to the sort value SMITH because there are two employees named Smith. Sort fields are displayed when their values change.

Example**Sorting Report Rows**

If you want to determine the sum of the total annual salaries in each department, you can issue the following request:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
ACROSS DEPARTMENT
END
```

The output is:

PAGE 1

DEPARTMENT MIS	PRODUCTION
\$108,002.00	\$114,282.00

Example**Sorting by Rows and Columns**

If you want to sum employees' current salaries across department and by job code, you can issue the following request:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
ACROSS DEPARTMENT
BY CURR_JOBCODE
END
```

The output is:

PAGE 1

CURR_JOBCODE	DEPARTMENT MIS	PRODUCTION
A01	.	\$9,500.00
A07	\$9,000.00	\$11,000.00
A15	.	\$26,862.00
A17	\$27,062.00	\$29,700.00
B02	\$18,480.00	\$16,100.00
B03	\$18,480.00	.
B04	\$21,780.00	\$21,120.00
B14	\$13,200.00	.

Selecting Records

When you generate a report, you may not want to include every record. Selecting records enables you to define a subset of the data source based on your criteria and then report on that subset. Your selection criteria can be as simple or complex as you wish.

For more information, see Chapter 5, *Selecting Records for Your Report*.

Example

Selecting Records

Suppose that you want to report on the salaries of employees earning more than \$20,000 a year. You can select the records for these employees using the WHERE phrase:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY LAST_NAME
WHERE CURR_SAL GT 20000
END
```

The output is:

PAGE 1

LAST_NAME	CURR_SAL
BANNING	\$29,700.00
BLACKWOOD	\$21,780.00
CROSS	\$27,062.00
IRVING	\$26,862.00
ROMANS	\$21,120.00

Showing Subtotals and Totals

To help interpret detailed information in a report, you can summarize numeric information using row and column totals, grand totals, and subtotals. You can use these summary lines to clarify or highlight information in numeric or matrix reports.

For more information, see Chapter 7, *Including Totals and Subtotals*.

Example**Showing Subtotals and Totals**

In your report on employee salaries, you might want to show each department's total salary expense.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY DEPARTMENT BY LAST_NAME
ON DEPARTMENT SUBTOTAL
END
```

The output is:

PAGE 1

DEPARTMENT	LAST_NAME	CURR_SAL
MIS	BLACKWOOD	\$21,780.00
	CROSS	\$27,062.00
	GREENSPAN	\$9,000.00
	JONES	\$18,480.00
	MCCOY	\$18,480.00
	SMITH	\$13,200.00
*TOTAL DEPARTMENT MIS		\$108,002.00
PRODUCTION	BANNING	\$29,700.00
	IRVING	\$26,862.00
	MCKNIGHT	\$16,100.00
	ROMANS	\$21,120.00
	SMITH	\$9,500.00
	STEVENS	\$11,000.00
*TOTAL DEPARTMENT PRODUCTION		\$114,282.00
TOTAL		\$222,284.00

Creating Temporary Fields

When you create a report, you are not limited to the fields that already exist in the data source. You can create new, temporary fields that are based on the values of existing fields. A temporary field can be independent of the report request. Once a temporary field is defined, you can use it in subsequent report requests within the same session. You can also create temporary fields within a particular report request. For details, see Chapter 6, *Creating Temporary Fields*.

In addition, you can define temporary fields in a Master File using the DEFINE attribute. For details, see the *Describing Data* manual.

Example

Creating Temporary Fields

If you want to include each employee's monthly salary in a report, and the annual salary is already stored in the CURR_SAL field, you could define a temporary field whose value is one-twelfth the value of CURR_SAL. You simply issue a DEFINE command prior to issuing the report request.

```
DEFINE FILE EMPLOYEE
MONTHLY_SAL = CURR_SAL/12;
END

TABLE FILE EMPLOYEE
PRINT EMP_ID AND MONTHLY_SAL AND CURR_SAL
END
```

The output is:

PAGE 1

EMP_ID	MONTHLY_SAL	CURR_SAL
071382660	916.67	\$11,000.00
112847612	1,100.00	\$13,200.00
117593129	1,540.00	\$18,480.00
119265415	791.67	\$9,500.00
119329144	2,475.00	\$29,700.00
123764317	2,238.50	\$26,862.00
126724188	1,760.00	\$21,120.00
219984371	1,540.00	\$18,480.00
326179357	1,815.00	\$21,780.00
451123478	1,341.67	\$16,100.00
543729165	750.00	\$9,000.00
818692173	2,255.17	\$27,062.00

Including Display Fields in a Report Request

The maximum number of display fields you can include in a report request is approximately 1024 (495 for MATCH requests). However, when adding fields to a request it is important to be aware that the allowable number of fields includes all named fields, whether printed or not, including data source fields, temporary fields (virtual fields and calculated values), certain internal fields (for example, TABPAGENO), and fields used in headings and footings. The total does not include sort fields.

This field limit is also affected by the combined length of fields in the request: that is, the field limit represents the maximum number of fields allowed when each field has the smallest length possible (A4 ACTUAL). Longer field lengths reduce the total number of printable fields.

When you create a report, the fields specified in the request are stored in a 32K (3956 bytes for MATCH requests) data area. The capacity of the data area is affected by a number of factors:

- Every field is rounded up to a full word boundary (a multiple of 4).
- Every field is associated with a four-byte counter field, which affects the total number of bytes in this data area.
- Field prefixes and formatting options impact the available data area.

If the combined length of the display fields in the data area exceeds the maximum capacity, an error message displays. To correct the problem, adjust the number or lengths of the fields in the request.

Referring to Fields in a Report Request

When creating a report, you refer to fields in several parts of the request—for example, in display commands (PRINT, SUM, etc.), in sort phrases (BY, ACROSS), and in selection criteria (WHERE, WHERE TOTAL, IF).

Several methods are available for referring to a field. You can:

- Refer to individual fields by using the alias specified in the Master File, referring to the name defined in the Master File, or using the shortest unique truncation of the field name or alias. For details, see *Referring to an Individual Field* on page 1-14.
- Refer to fields using long and qualified field names. For details, see *Referring to Fields Using Long and Qualified Field Names* on page 1-14.
- Refer to all fields in a segment using only one field name. For details, see *Referring to All of the Fields in a Segment* on page 1-16.

You can also view a list of all the fields that are included in the currently active data source, or a specified Master File. For details, see *Displaying a List of Field Names* on page 1-16 and *Listing Field Names, Aliases, and Format Information* on page 1-17.

Referring to an Individual Field

You can refer to an individual field in any one of the following ways:

- Using the field name defined in the Master File.
- Using the alias (the field name's synonym) defined in the Master File.
- Using the shortest unique truncation of the field name or the alias. When a truncation is used, it must be unique; if it is not unique, an error message is displayed.

Example

Referring to an Individual Field

In the following requests, DEPARTMENT is the complete field name, DPT is the alias, and DEP is a unique truncation of DEPARTMENT. All these examples produce the same output.

```
1. TABLE FILE EMPLOYEE
   PRINT DEPARTMENT
   END

2. TABLE FILE EMPLOYEE
   PRINT DPT
   END

3. TABLE FILE EMPLOYEE
   PRINT DEP
   END
```

Note: If you use a truncation that is not unique, the following message will appear:

```
(FOC016) THE TRUNCATED FIELDNAME IS NOT UNIQUE : D
```

Referring to Fields Using Long and Qualified Field Names

Field names and aliases have a maximum length of 66 characters, including up to two qualifiers and qualification characters. The names you assign to temporary fields may also be up to 66 characters. However, text fields and indexed field names in Master Files are limited to 12 characters; although the aliases for text and indexed fields may be up to 66 characters. Field names are always displayed as column titles in reports, unless a TITLE attribute or an AS phrase is used to provide an alternative name. For related information see Chapter 9, *Customizing Tabular Reports*.

You may use the file name, segment name, or both as a qualifier for a specified field. This is useful when structures contain duplicate field names. All referenced field names and aliases may be qualified.

Syntax

How to Activate Long and Qualified Field Names

The SET FIELDNAME command enables you to activate long (up to 66 characters) and qualified field names.

```
SET FIELDNAME = fieldname
```

where:

fieldname

Specifies the activation status of long and qualified field names. Valid identifiers include:

NEWS specifies that 66-character and qualified field names are supported; the maximum length is 66 characters. NEW is the default value.

NOTRUNC supports the 66-character maximum; does not permit unique truncations of field names.

OLD specifies that 66-character and qualified field names are not supported; the maximum length is 12 characters. The limit may be different for some types of non-FOCUS data sources.

Example

Using a Qualified Field Name to Refer to a Field

```
EMPLOYEE.EMPINFO.EMP_ID
```

Is the fully-qualified name of the field EMP_ID in the EMPINFO segment of the EMPLOYEE file. The maximum of 66 characters includes the name of the field or alias, plus an eight-character maximum for field qualifiers (Master File name and segment name) and two delimiting characters (periods).

Reference

Usage Notes for Long and Qualified Field Names

? SET displays the current value of FIELDNAME. In addition, a Dialogue Manager variable called &FOCFIELDNAME is available. &FOCFIELDNAME may have a value of NEW, OLD, or NOTRUNC.

When the value of FIELDNAME is changed within a session, JOIN, and DEFINE commands are affected as follows:

- When you change from a value of OLD to a value of NEW, all JOIN and DEFINE commands are cleared.
- When you change from a value of OLD to NOTRUNC, all JOIN and DEFINE commands are cleared.
- When you change from a value of NEW to OLD, all JOIN and DEFINE commands are cleared.
- When you change from a value of NOTRUNC to OLD, all JOIN and DEFINE commands are cleared.

All other changes to the FIELDNAME value have no effect on JOIN and DEFINE commands.

For additional information about using qualified field names in report requests, see the *Describing Data* manual.

Referring to All of the Fields in a Segment

If you want to generate a report that displays all of a segment's fields, you can refer to the complete segment without specifying every field. You only need to specify one field in the segment—any field will do—prefixed with the SEG. operator.

Example

Referring to All Fields in a Segment

The segment PRODS01 in the GGPRODS Master File contains the PRODUCT_ID, PRODUCT_DESCRIPTION, VENDOR_CODE, VENDOR_NAME, PACKAGE_TYPE, SIZE, and UNIT_PRICE fields.

```
SEGMENT=PRODS01
FIELDNAME = PRODUCT_ID
FIELDNAME = PRODUCT_DESCRIPTION
FIELDNAME = VENDOR_CODE
FIELDNAME = VENDOR_NAME
FIELDNAME = PACKAGE_TYPE
FIELDNAME = SIZE
FIELDNAME = UNIT_PRICE
```

To write a report that includes data from every field in the segment, you can issue either of the following requests:

1.

```
TABLE FILE GGPRODS
PRINT PRODUCT_ID AND PRODUCT_DESCRIPTION AND VENDOR_CODE AND
VENDOR_NAME      AND PACKAGE_TYPE AND SIZE AND UNIT_PRICE
END
```
2.

```
TABLE FILE GGPRODS
PRINT SEG.PRODUCT_ID
END
```

Displaying a List of Field Names

If you want to see a list of all the fields that are included in the currently active data source, you can issue the ?F field name query.

This is useful if you need to refer to a list of field names, or need to check the spelling of a field name, without exiting from the request process. It will also show you the entire 66-character field name. More information on all of the query (?) commands appears in the *Developing Applications* manual.

Listing Field Names, Aliases, and Format Information

The ?FF query displays field name, alias, and format information for a specified Master File, grouped by segment. Like the ?F query, you may issue ?FF:

- From the command line.
- When entering a TABLE or GRAPH request online.

If your software supports MODIFY or FSCAN, you can also issue ?FF from these facilities.

Note:

- If duplicate field names match a specified string, the display includes the field name qualified by the segment name with both ?F and ?FF.
- Field names longer than 31 characters are truncated in the display, and a caret (>) is appended in the 32nd position to indicate that the field name is longer than the display.
- When issuing a request in the Terminal Operator Environment, the ?F query activates the Fields window. However, ?FF makes the Output window active.

Customizing a Report

There are two aspects of a successful report: the information presented and how it is presented. A report that identifies related groups of information and draws attention to important facts will be more effective than one that simply shows columns of data.

When you have selected the data that is going to be included in your report and how you want it to be displayed, you can then continue developing your report with custom formatting. There are many things you can add to your request in order to make your report more effective. You can:

- Add titles, headings, and footings, change column titles with the AS phrase, create headings and footings for different levels of the report—including each sort group, each page, and the entire report. For details, see Chapter 9, *Customizing Tabular Reports*.
- Change the format of a field and the justification of a column title. For details, see *Changing the Format of a Report Column* on page 1-19.
- Determine the width of a report column. For details, see *Determining the Width of a Report Column* on page 1-24.
- Dynamically control the display of subtotals, headings, and footings based on conditions you define. For details, see Chapter 9, *Customizing Tabular Reports*.
- Highlight a group of related information and separate it from other groups by inserting blank lines or dashes between each group. For details, see Chapter 9, *Customizing Tabular Reports*.

Example Customizing the Presentation

The following report incorporates many customization features, such as renaming column titles, creating headings and footings for sections of the report, and dynamically controlling the display of headings and footings.

```
TABLE FILE EMPLOYEE
HEADING CENTER
"Departmental Salary Report </1"
PRINT CURR_JOBCODE AS 'Job Code'
BY DEPARTMENT AS 'Department'
BY LAST_NAME AS 'Last Name'
BY CURR_SAL AS 'Current,Salary'
ON CURR_SAL SUBFOOT
"<13 *** WARNING: <LAST_NAME 's salary exceeds recommended guidelines."
WHEN CURR_SAL GT 27000;
ON DEPARTMENT SUBFOOT
"<13 Total salary expense for the <DEP dept is: <ST.CURR_SAL"
ON DEPARTMENT SKIP-LINE
END

PAGE      1
```

Departmental Salary Report			
Department	Last Name	Current Salary	Job Code
MIS	BLACKWOOD	\$21,780.00	B04
	CROSS	\$27,062.00	A17
	*** WARNING: CROSS 's salary exceeds recommended guidelines.		
	GREENSPAN	\$9,000.00	A07
	JONES	\$18,480.00	B03
	MCCOY	\$18,480.00	B02
	SMITH	\$13,200.00	B14
	Total salary expense for the MIS dept is:		\$108,002.00
PRODUCTION	BANNING	\$29,700.00	A17
	*** WARNING: BANNING 's salary exceeds recommended guidelines.		
	IRVING	\$26,862.00	A15
	MCKNIGHT	\$16,100.00	B02
	ROMANS	\$21,120.00	B04
	SMITH	\$9,500.00	A01
	STEVENS	\$11,000.00	A07
Total salary expense for the PRODUCTION dept is:		\$114,282.00	

Changing the Format of a Report Column

A field's format is defined in the Master File. You can, however, change the format of a report column. Column titles in a report can be left justified, right justified, or centered. By default, column titles for alphanumeric fields are left justified, and column titles for numeric and date fields are right justified.

For details see Chapter 9, *Customizing Tabular Reports*.

Example

Changing a Column's Format

The UNIT_PRICE field has a format of D7.2 as defined in the GGPRODS Master File. To add a floating dollar sign to the display, the field format can be redefined as follows:

```
TABLE FILE GGPRODS
PRINT UNIT_PRICE/D7.2M
END
```

The output is:

```
Unit
Price
-----
$58.00
$81.00
$76.00
$13.00
$17.00
$28.00
$26.00
$96.00
$125.00
$140.00
```

Example **Using Multiple Format Specifications**

The following request illustrates column title justification with a format specification, a BY field specification, and an AS phrase specification:

```
TABLE FILE CAR
PRINT MODEL/A10 STANDARD/A15/R AS 'RJUST,STANDARD' BY CAR/C
WHERE CAR EQ 'JAGUAR' OR 'TOYOTA'
END
```

The output is:

CAR	MODEL	RJUST STANDARD
JAGUAR	V12XKE AUT XJ12L AUTO	POWER STEERING RECLINING BUCKE WHITEWALL RADIA WRAP AROUND BUM 4 WHEEL DISC BR
TOYOTA	COROLLA 4	BODY SIDE MOLDI MACPHERSON STRU

Reference **Usage Notes for Changing Column Format**

- Each time you reformat a column, the field is counted twice against the limit for display fields in a single report. For details, see *Including Display Fields in a Report Request* on page 1-13.
- If you create an extract file from the report—that is, a HOLD, PCHOLD, SAVE, or SAVB file—the extract file will contain fields for both the original format and the redefined format, unless HOLDLIST=PRINTONLY. Extract files are described in Chapter 11, *Saving and Reusing Report Output*.
- Format redefinition may not be used on a field in a BY or ACROSS phrase or with SUM CNT.fieldname.
- When the size of a word in a text field instance is greater than the format of the text field in the Master File, the word wraps to a second line, and the next word begins on the same line.
- You may specify justification for display fields, BY fields, and ACROSS fields. For ACROSS fields, data values, not column titles, are justified as specified.
- For display commands only, the justification parameter may be combined with a format specification. The format specification may precede or follow the justification parameter.
- If a title is specified with an AS phrase or in the Master File, that title will be justified as specified in FORMAT.
- When multiple ACROSS fields are requested, justification is performed on the lowest ACROSS level only. All other justification parameters for ACROSS fields are ignored.

Field-Based Reformatting

Field-based reformatting allows you to apply different formats to each row in a single report column by using a field to identify the format that applies to each row. For example, you can use this technique to apply the appropriate decimal currency formats when each row represents a different country.

The field that contains the format specifications can be:

- A real field in the data source.
- A temporary field created with a `DEFINE` command.
- A `DEFINE` in the Master File.
- A `COMPUTE` command. If the field is created with a `COMPUTE` command, the command must appear in the request prior to using the calculated field for reformatting.

The field that contains the formats must be alphanumeric and be at least eight characters in length. Only the first eight characters are used for formatting.

The field-based format may specify a length longer than the length of the original field. However, if the new length is more than one-third larger than the original length, the report column width may not be large enough to hold the value (indicated by asterisks in the field).

You can apply a field-based format to any type of field. However, the new format must be compatible with the original format:

- A numeric field can be reformatted to any other numeric format with any edit format options.
- An alphanumeric field can be reformatted to a different length.
- Any date field can be reformatted to any other date format type.
- Any date-time field can be reformatted to any other date-time format.

If the field-based format is invalid or specifies an impermissible type conversion, the field displays with plus signs (+++++) on the report output.

Syntax

How to Define and Apply a Format Field

- With a DEFINE command:

```
DEFINE FILE filename  
format_field/A8 = expression;  
END
```

- In a Master File:

```
DEFINE format_field/A8 = expression; $
```

- In a request:

```
COMPUTE format_field/A8 = expression;
```

where:

format_field

Is the name of the field that contains the format for each row.

expression

Is the expression that assigns the format values to the format field.

Once the format field is defined, you can apply it in a report request:

```
TABLE FILE filename  
display fieldname/format_field[/just]  
END
```

where:

display

Is any valid display command.

fieldname

Is a field in the request to be reformatted.

format_field

Is the name of the field that contains the formats. If the name of the format field is the same as an explicit format, the explicit format will be used. For example, a field named I8 cannot be used for field-based reformatting because it will be interpreted as the explicit format I8.

just

Is a justification option, L, R, or C. The justification option can be placed before or after the format field, separated from the format by a slash.

Example**Displaying Different Decimal Places for Currency Values**

```
DEFINE FILE CAR
CFORMAT/A8 = DECODE COUNTRY('ENGLAND' 'D10.1' 'JAPAN' 'D10' ELSE 'D10.2');
END
```

```
TABLE FILE CAR
SUM SALES/CFORMAT/C DEALER_COST/CFORMAT
BY COUNTRY
END
```

The output is:

COUNTRY	SALES	DEALER_COST
-----	-----	-----
ENGLAND	12,000.0	37,853.0
FRANCE	.00	4,631.00
ITALY	30,200.00	41,235.00
JAPAN	78,030	5,512
W GERMANY	88,190.00	54,563.00

Reference**Usage Notes for Field-Based Reformatting**

- Field-based reformatting is supported for TABLE and TABLEF. It works with StyleSheets, joins, and for any type of data source.
- Field-based reformatting is not supported for MODIFY, Maintain, MATCH, GRAPH, RECAP, FOOTING, HEADING, or text fields.
- Although you can use a DEFINE or COMPUTE command to create the format field, you *cannot* apply a field-based format to a calculated or virtual field.
- Field-based reformatting cannot be used on a BY sort field. It does work with an ACROSS field.
- If a report column is produced using field-based reformatting, the format used for a total or subtotal of the column will be taken from the previous detail line.
- Explicit reformatting creates two display fields internally for each field that is reformatted. Field-based reformatting creates three display fields.
- Field-based reformatting works for alphanumeric fields in a HOLD file, although three fields will be stored in the file for each field that is reformatted. To prevent the extra fields from being propagated to the HOLD file, specify SET HOLDLIST=PRINTONLY.
- If the number of decimal places varies between rows, the decimal points will not be aligned in the report output.

Determining the Width of a Report Column

The width of a report column is set to the width of the column title, or the corresponding field display length, whichever is wider. You can change the width by reformatting the column or editing the title.

For example, the LAST_NAME field is defined with a format of A15 in the Master File, while its field name is only nine characters wide, so the LAST_NAME column in a report will be 15 characters wide.

Furthermore, two spaces are placed between columns on the printed report unless the report width is too wide, in which case one space is inserted between columns. If the report is still too wide, it will need to be paneled.

Note: The default spacing can be overridden by using IN or OVER. You can also use SET SPACES to control column spacing. For more information see Chapter 9, *Customizing Tabular Reports*.

Saving and Reusing Report Output

After generating a report request, there are many things you can do with it. You can create an output file using the HOLD, PCHOLD, SAVE, and SAVB commands. This allows you to store your report as a data source on which you can make additional queries. This capability is especially helpful for creating a subset of your data and for generating multi-step reports. You can also format the report output for:

- Display as (or in) a web page, as a printed document, or in a text document.
- Processing in another application, such as a spreadsheet, a database, a word processor, or a 3GL program.
- Distribution to another location, such as a PC or a browser.

For details, see Chapter 11, *Saving and Reusing Report Output*.

CHAPTER 2

Displaying Report Data

Topics:

- Displaying Individual Values
- Adding Values
- Counting Values
- Manipulating Display Fields With Prefix Operators
- Manipulating Display Field Values in a Sort Group

Reporting, at the simplest level, retrieves field values from a data source and displays those values. There are three ways to do this:

- List each field value (PRINT and LIST commands).
- Add all the values and display the sum (SUM command).
- Count all the values and display the quantity (COUNT command).

These four display commands—PRINT, LIST, SUM, and COUNT—are also known as verbs. These commands are flexible; you can report from several fields using a single command and include several different display commands in a single report request.

Syntax

How to Use Display Commands in a Request

`display [THE] fieldname1 [AND] [THE] fieldname2 ...`

OR

`display *`

where:

`display`

Is the PRINT, LIST, SUM, or COUNT command. WRITE and ADD are synonyms of SUM and can be substituted for it.

`fieldname`

Is the name of the field to be displayed in the report.

The maximum number of display fields your report can contain is determined by a combination of factors. For details, see Chapter 1, *Creating Tabular Reports*.

The fields appear in the report in the same order in which they are specified in the report request. For example, the report column for *fieldname1* will appear first, followed by the report column for *fieldname2*.

The field to be displayed is also known as the display field.

`AND`

Is optional; used to enhance readability. It can be used between any two field names and does not affect the report.

`THE`

Is optional; used to enhance readability. It can be used before any field name and does not affect the report.

`*`

Applies the display command to every field in the left path of the data source.

Displaying Individual Values

The display commands LIST and PRINT list the individual values of the fields you specify in your report request. LIST numbers the items in the report. PRINT does not number the items.

You can easily display all of the fields in the data source by specifying an asterisk (*) wildcard instead of a specific field name, as described in *Displaying All Fields* on page 2-5.

For all PRINT and LIST requests, the number of records retrieved and the number of lines displayed are the same. In addition, there is no order to the report rows. The PRINT and LIST commands simply display all the values of the selected fields found in the data source, in the order in which they are accessed. The order in which data is displayed may be affected by the AUTOPATH setting. For more information, see Chapter 15, *Improving Report Processing* and the documentation on SET parameters in the *Developing Applications* manual.

In general, when using PRINT or LIST, the order of the values displayed in the report depends on whether the field is a key field or not, as described in the *Describing Data* manual.

Alternatively, you can sort the values using the BY or ACROSS sort phrases. When LIST is used in a request that includes a sort phrase, the list counter is reset to 1 every time the value in the outermost sort field changes. See Chapter 4, *Sorting Tabular Reports*, for more information on sorting.

Example **Displaying Individual Field Values**

To display the values of individual fields, use the PRINT command. The following request displays the values of two fields—LAST_NAME and FIRST_NAME— for all employees.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
END
```

The output is:

PAGE	1
LAST_NAME	FIRST_NAME
-----	-----
STEVENS	ALFRED
SMITH	MARY
JONES	DIANE
SMITH	RICHARD
BANNING	JOHN
IRVING	JOAN
ROMANS	ANTHONY
MCCOY	JOHN
BLACKWOOD	ROSEMARIE
MCKNIGHT	ROGER
GREENSPAN	MARY
CROSS	BARBARA

Example **Listing Records**

To number the records in a report, use the LIST command.

```
TABLE FILE EMPLOYEE
LIST LAST_NAME AND FIRST_NAME
END
```

The output is:

PAGE	1		
LIST	LAST_NAME		FIRST_NAME
----	-----		-----
1	STEVENS		ALFRED
2	SMITH		MARY
3	JONES		DIANE
4	SMITH		RICHARD
5	BANNING		JOHN
6	IRVING		JOAN
7	ROMANS		ANTHONY
8	MCCOY		JOHN
9	BLACKWOOD		ROSEMARIE
10	MCKNIGHT		ROGER
11	GREENSPAN		MARY
12	CROSS		BARBARA

Displaying All Fields

You can easily display all of the fields in the left path of the data source by specifying an asterisk (*) wildcard instead of a specific field name. For additional information about Master File structures and segment paths, including left paths and short paths, see the *Describing Data* manual.

Example

Displaying All Fields

The following request produces a report displaying all of the fields in the EDUCFILE data source.

```
TABLE FILE EDUCFILE
LIST *
END
```

The output is:

LIST	COURSE_CODE	COURSE_NAME	DATE_ATTEND	EMP_ID
---	-----	-----	-----	-----
1	101	FILE DESCRPT & MAINT	83/01/04	212289111
2	101	FILE DESCRPT & MAINT	82/05/25	117593129
3	101	FILE DESCRPT & MAINT	82/05/25	071382660
4	101	FILE DESCRPT & MAINT	81/11/15	451123478
5	101	FILE DESCRPT & MAINT	81/11/15	112847612
6	102	BASIC REPORT PREP NON-PROG	82/07/12	326179357
7	103	BASIC REPORT PREP FOR PROG	83/01/05	212289111
8	103	BASIC REPORT PREP FOR PROG	82/05/26	117593129
9	103	BASIC REPORT PREP FOR PROG	81/11/16	112847612
10	104	FILE DESC & MAINT NON-PROG	82/07/14	326179357
11	106	TIMESHARING WORKSHOP	82/07/15	326179357
12	202	WHAT'S NEW IN FOCUS	82/10/28	326179357
13	301	DECISION SUPPORT WORKSHOP	82/09/03	326179357
14	107	BASIC REPORT PREP DP MGRS	82/08/02	818692173
15	302	HOST LANGUAGE INTERFACE	82/10/21	818692173
16	108	BASIC RPT NON-DP MGRS	82/10/10	315548712
17	108	BASIC RPT NON-DP MGRS	82/08/24	119265415
18	201	ADVANCED TECHNIQUES	82/07/26	117593129
19	203	FOCUS INTERNALS	82/10/28	117593129

Displaying the Structure of a Multi-Path Data Source

When using display commands, it is important to understand the structure of the data source and the relationship between segments since these factors will affect your results. You can use the CHECK command's PICTURE option to display a diagram of the data source structure defined by the Master File.

Example

Displaying the Structure of a Multi-Path Data Source

To display the structure of the EMPLOYEE data source, which is joined to the JOBFIL and EDUCFILE data sources, issue the following command:

```
CHECK FILE EMPLOYEE PICTURE RETRIEVE
```

This command adds the numbers that display at the top left of each segment, indicating the retrieval order of the segments, and generates the following picture. Note that a unique segment such as FUNDTRAN is treated as a logical addition to the parent segment for retrieval. FUNDTRAN and SECSEG are unique segments and are, therefore, treated as part of their parents.

```

check file employee picture retrieve
NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS= 11 ( REAL=  6 VIRTUAL=  5 )
NUMBER OF FIELDS=  34 INDEXES=  0 FILES=  3
TOTAL LENGTH OF ALL FIELDS= 365
SECTION 01
      RETRIEVAL VIEW OF FOCUS      FILE EMPLOYEE ON 12/29/93 AT 14.42.18

      EMPINFO
01      S1
*****
*EMP_ID      **
*LAST_NAME   **
*FIRST_NAME  **
*HIRE_DATE   **
*           **
*****

*****
I
I
I
I FUNDTRAN
02      I U
*****
*BANK_NAME   *
*BANK_CODE   *
*BANK_ACCT   *
*EFFECT_DATE *
*           *
*****
I
I
I
I
I PAYINFO      I ADDRESS      I SALINFO      I ATTNDSEG
03      I SH1   07      I S1     08      I SH1     10      I KM
*****
*DAT_INC      ** *TYPE      ** *PAY_DATE      ** :DATE_ATTEND **
*PCT_INC      ** *ADDRESS_LN1 ** *GROSS      ** :EMP_ID      **:K
*SALARY      ** *ADDRESS_LN2 ** *           ** :           **:
*JOBCODE      ** *ADDRESS_LN3 ** *           ** :           **:
*           ** *           ** *           ** :           **:
*****
*****
I
I
I
I
I JOBSEG
04      I KU
*****
:JOBCODE      :K
:JOB_DESC      :
:           :
:           :
:           :
*****
I JOBFIL
I
I
I
I
I SECSEG
05      I KLU
*****
:SEC_CLEAR      :
:           :
:           :
:           :
*****
I JOBFIL
I
I
I
I
I SKILLSEG
06      I KL
*****
:SKILLS      :
:SKILL_DESC    :
:           :
:           :
:           :
*****

```

Example

Displaying Fields From a Multi-Path Data Source

The following request produces a report displaying all of the fields on the left path of the EMPLOYEE data source.

```
TABLE FILE EMPLOYEE
PRINT *
END
```

Due to the size of the report the above request produces, we will list only the fields for which all instances will be printed. In the report, these fields would be displayed from left to right starting with EMP_ID.

```
EMP_ID
LAST_NAME
FIRST_NAME
HIRE_DATE
DEPARTMENT
CURR_SAL
CURR_JOBCODE
ED_HRS
BANK_NAME
BANK_CODE
BANK_ACCT
EFFECT_DATE
DAT_INC
PCT_INC
SALARY
JOBCODE
JOBDESC
SEC_CLEAR
SKILLS
SKILL_DESC
```

Each field in this list appears in segments on the left path of the EMPLOYEE data source. To view the structure of the EMPLOYEE data source, see *Displaying the Structure of a Multi-Path Data Source* on page 2-6.

Tip:

In some environments the following warning is displayed whenever you use PRINT * with a multi-path data source, to remind you that PRINT * will display only the left path.

```
(FOC757) WARNING. YOU REQUESTED PRINT * OR COUNT * FOR A MULTI-PATH
FILE
```

Adding Values

SUM, WRITE, and ADD sum the values of a numeric field. The three commands are synonyms; they can be used interchangeably, and every reference to SUM in this manual also refers to WRITE and ADD.

When you use SUM, multiple records are read from the data source, but only one summary line is produced. If you use SUM with a non-numeric field—such as an alphanumeric, text, or date field—SUM will not add the values; instead, it will display the last value retrieved from the data source.

For SUM, WRITE, and ADD syntax see *How to Use Display Commands in a Request* on page 2-2.

Example

Adding Values

This request adds all the values of the field CURR_SAL:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
END
```

The output is:

NUMBER OF RECORDS IN TABLE= 12 LINES= 1

PAGE 1

```
      CURR_SAL
      -----
$222,284.00
```

The number of lines in the report is less than the number of records from the data source. It took a total of 12 records to get the results in the report, but only one summary line is displayed.

Example **Adding Non-Numeric Values**

This request attempts to add non-numeric fields. Any request for aggregation on non-numeric data returns the last record retrieved from the data source.

```
TABLE FILE EMPLOYEE
SUM LAST_NAME AND FIRST_NAME
END
```

The output is:

NUMBER OF RECORDS IN TABLE= 12 LINES= 1

PAGE 1

LAST_NAME	FIRST_NAME
-----	-----
CROSS	BARBARA

Note that any request for aggregation on a date format field also returns the last record retrieved from the data source.

Tip:

If you are using the external sorting product DFSORT, you can set the SUMPREFIX parameter to FST or LST to control the sort order. For details, see Chapter 4, *Sorting Tabular Reports*.

Counting Values

The COUNT command counts the number of instances that exist for a specified field. The COUNT command is particularly useful combined with the BY phrase, which is discussed in Chapter 4, *Sorting Tabular Reports*.

COUNT counts the instances of data in a data source, not the distinct values.

Note: By default, a COUNT field is a five-digit integer. You can reformat it using the COMPUTE command and change its field length using the SET COUNTWIDTH command. For details about the COMPUTE command, see Chapter 6, *Creating Temporary Fields*. For information about SET COUNTWIDTH, see the *Developing Applications* manual.

Example

Counting Values

To determine how many employees are in the EMPLOYEE data source, you can count the instances of EMP_ID, the employee identification number.

```
TABLE FILE EMPLOYEE
COUNT EMP_ID
END
```

The output is:

NUMBER OF RECORDS IN TABLE= 12 LINES= 1

PAGE 1

```
EMP_ID
COUNT
-----
12
```

Example

Counting Values With a Sort Phrase

To count the instances of EMP_ID for each department, use this request:

```
TABLE FILE EMPLOYEE
COUNT EMP_ID
BY DEPARTMENT
END
```

The output indicates that of the 12 EMP_IDs in the data source, six are from the MIS department and six are from the PRODUCTION department:

```
EMP_ID
DEPARTMENT COUNT
-----
MIS 6
PRODUCTION 6
```

Example **Counting Instances of Data**

The following example counts the instances of data in the LAST_NAME, DEPARTMENT, and JOBCODE fields in the EMPLOYEE data source.

```
TABLE FILE EMPLOYEE
COUNT LAST_NAME AND DEPARTMENT AND JOBCODE
END
```

The output is:

LAST_NAME	DEPARTMENT	JOBCODE
COUNT	COUNT	COUNT
-----	-----	-----
12	12	19

The EMPLOYEE data source contains data on 12 employees, with one instance for each LAST_NAME. While there are only two values for DEPARTMENT, there are 12 instances of the DEPARTMENT field because each employee works for one of the two departments. Similarly, there are 19 instances of the JOBCODE field because employees can have more than one job code during their employment.

Counting Segment Instances

You can easily count the instances of the lowest segment in the left path of a data source by specifying an asterisk (*) wildcard instead of a specific field name. In a single-segment data source, this effectively counts all instances in the data source.

COUNT * accomplishes this by counting the values of the first field in the segment. Instances with a missing value in the first field are not counted (when SET MISSING=ON).

Segment instances in short paths are not counted by COUNT *, regardless of the value of the ALL parameter of the SET command.

For more information about missing values, short paths, and the SET ALL command, see Chapter 12, *Handling Records With Missing Field Values*.

Example

Counting Segments From a Multi-Path Data Source

The following request counts the number of instances of the SKILLSEG segment of the EMPLOYEE data source.

```
TABLE FILE EMPLOYEE
COUNT *
END
```

The output is:

```
PAGE      1

COUNT *
COUNT
-----
      19
```

COUNT * counts the number of instances of the SKILLSEG segment, which is the lowest segment in the left path of the EMPLOYEE data source structure (that is, the EMPLOYEE data source joined to the JOBFILE and EDUCFILE data sources). You can see a picture of the path structure in *Displaying the Structure of a Multi-Path Data Source* on page 2-6.

Tip:

In some environments the following warning is displayed if you use COUNT * with a multi-path data source (such as EMPLOYEE in the above example):

(FOC757) WARNING. YOU REQUESTED PRINT * OR COUNT * FOR A MULTI-PATH FILE

Expanding Byte Precision for COUNT and LIST

The COUNT and LIST commands can optionally be expanded from 5 to 9 characters on display. This internally reformats COUNT and LIST from I5 to I9.

If the number of records retrieved for a field exceeds 99,999 (5 bytes), asterisks are displayed in the report to indicate an overflow condition. You can increase the display to allow a COUNT or LIST as large as 999,999,999 (9 bytes).

Syntax

How to Set the Precision for COUNT and LIST

SET COUNTWIDTH = {[OFF](#)|[ON](#)}

where:

[OFF](#)

Displays five characters (bytes) for COUNT and LIST counter values. Asterisks are displayed if the number of records retrieved for a field exceeds 5 characters. OFF is the default.

[ON](#)

Displays up to nine characters (bytes) for COUNT and LIST counter values. Asterisks are displayed if the value exceeds 9 characters.

Example

Setting Precision for COUNT and LIST

The following example shows the COUNT command with SET COUNTWIDTH = OFF:

```
TABLE FILE filename
COUNT Fldxx
BY Fldyy
END

          FLDxx

Fldyy    COUNT
value    *****
```

The following example shows the COUNT command with SET COUNTWIDTH = ON:

```
TABLE FILE filename
COUNT Fldxx
BY Fldyy
END

          FLDxx

Fldyy    COUNT
value    999999999
```

Note: This feature will affect the width of a report when COUNTWIDTH is set to ON. Calculating the width of a report will now require an additional four display positions for each COUNT and LIST column.

Manipulating Display Fields With Prefix Operators

You can use prefix operators to perform calculations directly on the values of fields. For example, you can:

- Calculate the average of field values.
- Determine the minimum or maximum of field values.
- Calculate the percent value of field and count values.
- Determine the square and average of squared values.
- Retrieve the first or last record.
- Sum and count numeric values.
- Count the number of distinct (unique) values.

For a list of prefix operators and their functions see *Functions You Can Perform With Prefix Operators* on page 2-16.

Syntax

How to Use Prefix Operators

Each prefix operator is applied to a single field and affects only that field.

`{SUM|COUNT} prefix.fieldname`

where:

`prefix`

Is any prefix operator.

`fieldname`

Is the name of the field to be displayed in the report.

Reference

Usage Notes for Prefix Operators

- Because PRINT and LIST display individual field values, not an aggregate value, they are not used with prefix operators, except TOT.
- To sort by the results of a prefix command, use the phrase BY TOTAL to aggregate and sort numeric columns simultaneously. For details, see Chapter 4, *Sorting Tabular Reports*.
- The WITHIN phrase is very useful when using prefixes. For details, see *Manipulating Display Field Values in a Sort Group* on page 2-29.
- You can use the results of prefix operators in COMPUTE commands.
- With the exception of CNT. and PCT.CNT., resulting values have the same format as the field against which the prefix operation was performed.
- Text fields can only be used with the FST., LST., and CNT. prefix operators.

Reference

Functions You Can Perform With Prefix Operators

Prefix	Function
ASQ.	Computes the average sum of squares for standard deviation in statistical analysis.
AVE.	Computes the average value of the field.
CNT.	Counts the number of occurrences of the field.
CNT.DST.	Counts the number of distinct values within a field when using -REMOTE. For other modes of operation, this behaves like CNT.
DST.	Determines the total number of distinct values in a single pass of a data source.
FST.	Generates the first physical instance of the field. Can be used with numeric or text fields.
LST.	Generates the last physical instance of the field. Can be used with numeric or text fields.
MAX.	Generates the maximum value of the field.
MIN.	Generates the minimum value of the field.
PCT.	Computes a field's percentage based on the total values for the field. The PCT operator can be used with detail as well as summary fields
PCT.CNT.	Computes a field's percentage based on the number of instances found.
RPCT.	Computes a field's percentage based on the total values for the field across a row.
SUM.	Sums the number of occurrences of the field.
TOT.	Counts the occurrences of the field for use in a heading (includes footings, subheads, and subfoots).

Averaging Values of a Field

The AVE. prefix computes the average value of a particular field. The computation is performed at the lowest sort level of the display command. It is computed as the sum of the field values within a sort group divided by the number of records in that sort group. If the request does not include a sort phrase, AVE. calculates the average for the entire report.

Example

Averaging Values of a Field

This request calculates the average number of education hours spent in each department.

```
TABLE FILE EMPLOYEE
SUM AVE.ED_HRS BY DEPARTMENT
END
```

The output is:

PAGE 1

DEPARTMENT	AVE ED_HRS
MIS	38.50
PRODUCTION	20.00

Averaging the Sum of Squared Fields

The ASQ. prefix computes the average sum of squares, which is a component of the standard deviation in statistical analysis.

$$\left(\sum_{i=1}^n x_i^2 \right) / n$$

The ASQ. prefix is only supported for double-precision fields.

If the field format is packed, ASQ. is not supported.

If the field format is integer and you get a large set of numbers, the ASQ. result may be negative as a result of field overflow.

Example Averaging the Sum of Squared Fields

This request calculates the sum and the sum of squared fields for the DELIVER_AMT field.

```
TABLE FILE SALES
SUM DELIVER_AMT AND ASQ.DELIVER_AMT
BY CITY
END
```

The output is:

PAGE 1

CITY	DELIVER_AMT	ASQ DELIVER_AMT
NEW YORK	190	807
NEWARK	60	900
STAMFORD	430	3637
UNIONDALE	80	1600

Calculating Maximum and Minimum Field Values

The prefixes MAX. and MIN. produce the maximum and minimum values, respectively, within a sort group. If the request does not include a sort phrase, MAX. and MIN. produce the maximum and minimum values for the entire report.

Example Calculating Maximum and Minimum Field Values

This report request calculates the maximum and minimum values of SALARY.

```
TABLE FILE EMPLOYEE
SUM MAX.SALARY AND MIN.SALARY
END
```

The output is:

PAGE 1

MAX SALARY	MIN SALARY
\$29,700.00	\$8,650.00

Calculating Column and Row Percents

For each individual value in a column, PCT. calculates what percentage that field makes up of the column's total value. You can control how values are distributed down the column by sorting the column using the BY phrase. The new column of percentages has the same format as the original field.

You can also determine percentages for row values; for each individual value in a row that has been sorted using the ACROSS phrase, the RPCT. operator calculates what percentage it makes up of the row's total value. The percentage values have the same format as the original field.

Example

Calculating Column Percents

To calculate each employee's share of education hours, issue the following request:

```
TABLE FILE EMPLOYEE
SUM ED_HRS PCT.ED_HRS BY LAST_NAME
ON TABLE COLUMN-TOTAL
END
```

The output is:

LAST_NAME	ED_HRS	PCT ED_HRS
-----	-----	-----
BANNING	.00	.00
BLACKWOOD	75.00	21.37
CROSS	45.00	12.82
GREENSPAN	25.00	7.12
IRVING	30.00	8.55
JONES	50.00	14.25
MCCOY	.00	.00
MCKNIGHT	50.00	14.25
ROMANS	5.00	1.42
SMITH	46.00	13.11
STEVENS	25.00	7.12
 TOTAL	 351.00	 100.00

Since PCT. and RPCT take the same format as the field, the column may not total exactly 100 because of the nature of floating point arithmetic.

Example Calculating Row Percents

The following request calculates the total units sold for each product (UNIT_SOLD column) and the percentage that total makes up in relation to the sum of all products sold (RPCT.UNIT_SOLD column) in each city.

```
TABLE FILE SALES
SUM UNIT_SOLD RPCT.UNIT_SOLD
BY PROD_CODE
ACROSS CITY
END
```

Because the full report is too wide to display, a representative portion of the output is shown here:

PAGE 1

PROD_CODE	CITY NEW YORK		NEWARK		STAMFORD		UNIONDALE	
	RPCT		RPCT		RPCT		RPCT	
	UNIT_SOLD	UNIT_SOLD	UNIT_SOLD	UNIT_SOLD	UNIT_SOLD	UNIT_SOLD	UNIT_SOLD	UNIT_SOLD
B10	30	29	13	12	60	58	.	.
B12	.	.	29	42	40	57	.	.
B17	20	40	.	.	29	59	.	.
B20	15	37	25	62
C13	25	100	.	.
C17	12	100
C7	45	52	40	47
D12	20	42	.	.	27	57	.	.
E1	30	100
E2	80	100	.	.
E3	35	33	.	.	70	66	.	.

Because UNIT_SOLD has an integer format, the columns created by RPCT. also have integer (I) formats. Therefore, individual percentages are truncated and the total percentage is less than 100%. If you require precise totals, redefine the field with a format that declares decimal places (D, F).

Producing a Direct Percent of a Count

When counting occurrences in a file, a common reporting need is determining the relative percentages of all found instances, which each count of a data value represents. You can do this, for columns only, with the following syntax:

```
PCT.CNT.fieldname
```

The format is a decimal value of six digits with two decimal places (F6.2).

Example

Producing a Direct Percent of a Count

This request illustrates the relative percentage of the values in the EMP_ID field for each department.

```
TABLE FILE EMPLOYEE
SUM PCT.CNT.EMP_ID
BY DEPARTMENT
END
```

The output is:

PAGE 1

DEPARTMENT	PCT.CNT EMP_ID
MIS	50.00
PRODUCTION	50.00

Aggregating and Listing Unique Values

The distinct prefix operator (DST.) may be used to aggregate and list unique values of any data source field. Similar in function to the SQL COUNT, SUM, and AVG(DISTINCT col) column functions, it permits you to determine the total number of distinct values in a single pass of the data source.

The DST. operator can be used with the SUM, PRINT or COUNT commands, and also in conjunction with the aggregate prefix operators SUM., CNT., and AVE.

Syntax

How to Use the Distinct Operator

`{command} DST.fieldname`

or

`SUM [operator].DST.fieldname`

where:

`command`

Is SUM, PRINT, or COUNT.

`DST.`

Indicates the distinct operator.

`fieldname`

Indicates the display-field object or field name.

`operator`

Indicates SUM., CNT., or AVE.

Example

Using the Distinct Operator

The procedure requesting a count of unique ED_HRS values is either:

```
TABLE FILE EMPLOYEE
SUM CNT.DST.ED_HRS
END
```

or

```
TABLE FILE EMPLOYEE
COUNT DST.ED_HRS
END
```

The output is:

```
COUNT
DISTINCT
ED_HRS
-----
          9
```

Notice that the count excludes the second records for values 50.00, 25.00, and .0 resulting in nine unique ED_HRS values.

When used with PRINT, DST. acts in the same manner as a BY phrase. It can be attached to several PRINT display fields, but not more than 32 (the current limit for BY sort fields).

DST. display fields must precede all non-distinct display fields named by the PRINT command. If this rule is not observed, the following error is displayed:

```
(FOC1855) DISTINCT FIELDS MUST PRECEDE THE NONDISTINCT ONES
```

Reference

Distinct Operator Limitations

- The following error occurs if you use the prefix operators CNT., SUM., and AVE. with any other display command:
(FOC1853) CNT/SUM/AVE.DST CAN ONLY BE USED WITH AGGREGATION VERBS
- The following error occurs if you use DST. in a MATCH or TABLEF command:
(FOC1854) THE DST OPERATOR IS ONLY SUPPORTED IN TABLE REQUESTS
- The following error occurs if you code more than one DST. operator for the SUM command:
(FOC1856) ONLY ONE DISTINCT FIELD IS ALLOWED IN AGGREGATION
- The following error occurs if you reformat a BY field (when used with the PRINT command, the DST.*fieldname* becomes a BY field):
(FOC1862) REFORMAT DST.FIELD IS NOT SUPPORTED WITH PRINT
- The following error occurs if you use the DST. operator in an ACROSS or FOR phrase:
(FOC1864) THE DST OPERATOR IS NOT SUPPORTED FOR ACROSS OR FOR
- The following error occurs if you use a multi-verb request, SUM DST.*fieldname* BY *field* PRINT *fld* BY *fld* (a verb object operator used with the SUM command must be at the lowest level of aggregation):
(FOC1867) DST OPERATOR MUST BE AT THE LOWEST LEVEL OF AGGREGATION
- The DST. operator may not be used as part of a HEADING or a FOOTING.
- TABLE requests that contain the DST. operator are not candidates for AUTOTABLEF.

Retrieving First and Last Records

FST. is a prefix that displays the first retrieved record selected for a given field. LST. displays the last retrieved record selected for a given field.

When using the FST. and LST. prefix operators, it is important to understand how your data source is structured.

- If the record is in a segment with values organized from lowest to highest (segment type S1), the first logical record that the FST. prefix operator retrieves is the lowest value in the set of values. The LST. prefix operator would, therefore, retrieve the highest value in the set of values.
- If the record is in a segment with values organized from highest to lowest (segment type SH1), the first logical record that the FST. prefix operator retrieves is the highest value in the set of values. The LST. prefix operator would, therefore, retrieve the lowest value in the set of values.

For more information on segment types and file design, see the *Describing Data* manual. If you wish to reorganize the data in the data source or restructure the data source while reporting, see Chapter 15, *Improving Report Processing*.

Example

Retrieving the First Record

The following request retrieves the first logical record in the EMP_ID field:

```
TABLE FILE EMPLOYEE
SUM FST.EMP_ID
END
```

The output is:

```
PAGE      1

FST
EMP_ID
-----
071382660
```

Example**Segment Types and Retrieving Records**

The EMPLOYEE data source contains the DEDUCT segment, which orders the fields DED_CODE and DED_AMT from lowest value to highest value (segment type of S1). The DED_CODE field indicates the type of deduction, such as CITY, STATE, FED, and FICA. The following request retrieves the first logical record for DED_CODE for each employee:

```
TABLE FILE EMPLOYEE
SUM FST.DED_CODE
BY EMP_ID
END
```

The output is:

PAGE 1

EMP_ID	FST DED_CODE
-----	-----
071382660	CITY
112847612	CITY
117593129	CITY
119265415	CITY
119329144	CITY
123764317	CITY
126724188	CITY
219984371	CITY
326179357	CITY
451123478	CITY
543729165	CITY
818692173	CITY

Note, however, the command SUM LST.DED_CODE would have retrieved the last logical record for DED_CODE for each employee.

If the record is in a segment with values organized from highest to lowest (segment type SH1), the first logical record that the FST. prefix operator retrieves is the highest value in the set of values. The LST. prefix operator would, therefore, retrieve the lowest value in the set of values.

For example, the EMPLOYEE data source contains the PAYINFO segment, which orders the fields JOBCODE, SALARY, PCT_INC, and DAT_INC from highest value to lowest value (segment type SH1). The following request retrieves the first logical record for SALARY for each employee:

```
TABLEF FILE EMPLOYEE
SUM FST.SALARY
BY EMP_ID
END
```

The output is:

PAGE 1

EMP_ID	FST SALARY
071382660	\$11,000.00
112847612	\$13,200.00
117593129	\$18,480.00
119265415	\$9,500.00
119329144	\$29,700.00
123764317	\$26,862.00
126724188	\$21,120.00
219984371	\$18,480.00
326179357	\$21,780.00
451123478	\$16,100.00
543729165	\$9,000.00
818692173	\$27,062.00

However, the command SUM LST.SALARY would have retrieved the last logical record for SALARY for each employee.

Summing and Counting Values

You can count occurrences and summarize values all in one display command using the prefixes CNT., SUM., and TOT. Just like the COUNT command, CNT. counts the occurrences of the one field it prefixes; just like the SUM command, SUM. sums the values of a particular field. TOT. counts the occurrences of the field for use in a heading (includes footings, subheads, and subfoots).

Example

Counting Values With CNT

The following request counts the occurrences of PRODUCT_ID and sums the value of UNIT_PRICE.

```
TABLE FILE GGPRODS
SUM CNT.PRODUCT_ID AND UNIT_PRICE
END
```

The output is:

PRODUCT_ID	Unit
COUNT	Price
-----	-----
10	660.00

Example

Summing Values With SUM

The following request counts the occurrences of PRODUCT_ID and sums the value of UNIT_PRICE.

```
TABLE FILE GGPRODS
COUNT PRODUCT_ID AND SUM.UNIT_PRICE
END
```

The output is:

PRODUCT_ID	Unit
COUNT	Price
-----	-----
10	660.00

Example

Counting Values With TOT

The following request uses the TOT prefix operator to show the total of current salaries for all employees.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
BY DEPARTMENT
ON TABLE SUBFOOT
"Total salaries equal: <TOT.CURR_SAL"
END
```

The output is:

DEPARTMENT	LAST_NAME
-----	-----
MIS	SMITH
	JONES
	MCCOY
	BLACKWOOD
	GREENSPAN
	CROSS
PRODUCTION	STEVENS
	SMITH
	BANNING
	IRVING
	ROMANS
	MCKNIGHT
Total salaries equal:	\$222,284.00

Manipulating Display Field Values in a Sort Group

You can use the WITHIN phrase to manipulate a display field's values as they are aggregated within a sort group. This technique can be used with a prefix operator to perform calculations on a specific aggregate field rather than a report column. In contrast, the SUM and COUNT commands aggregate an entire column.

The WITHIN phrase requires a BY phrase and/or an ACROSS phrase. A maximum of two WITHIN phrases can be used per display command. If one WITHIN phrase is used, it must act on a BY phrase. If two WITHIN phrases are used, the first must act on a BY phrase and the second on an ACROSS phrase.

Syntax

How to Use WITHIN to Manipulate Display Fields

```
{SUM|COUNT} display_field WITHIN by_sort_field [WITHIN across_sort_field]  
BY by_sort_field [ACROSS across_sort_field]
```

where:

display_field

Is the object of a SUM or COUNT display command.

by_sort_field

Is the object of a BY phrase.

across_sort_field

Is the object of an ACROSS phrase.

Reference

Usage Notes for WITHIN

- You can use up to 64 fields in a display command with the WITHIN phrase.
- You can also use the syntax WITHIN TABLE, which allows you to return the original value within a request command.
- The WITHIN TABLE command can also be used when an ACROSS phrase is needed without a BY phrase. Otherwise, a single WITHIN phrase requires a BY phrase.

Example Summing Values Within Sort Groups

The following report shows the units sold and the percent of units sold for each product within store and within the table:

```
TABLE FILE SALES
SUM UNIT_SOLD AS 'UNITS'
AND PCT.UNIT_SOLD AS 'PCT,SOLD,WITHIN,TABLE'
AND PCT.UNIT_SOLD WITHIN STORE_CODE AS 'PCT,SOLD,WITHIN,STORE'
BY STORE_CODE SKIP-LINE BY PROD_CODE
END
```

The output is:

PAGE		1		
	STORE_CODE	PROD_CODE	UNITS	PCT SOLD WITHIN TABLE
	-----	-----	-----	-----
	K1	B10	13	1
		B12	29	4
	14B	B10	60	8
		B12	40	5
		B17	29	4
		C13	25	3
		C7	45	6
		D12	27	3
		E2	80	11
		E3	70	10
				18
PAGE		2		
	STORE_CODE	PROD_CODE	UNITS	PCT SOLD WITHIN TABLE
	-----	-----	-----	-----
	14Z	B10	30	4
		B17	20	2
		B20	15	2
		C17	12	1
		D12	20	2
		E1	30	4
		E3	35	5
				21
	77F	B20	25	3
		C7	40	5
				38
				61

CHAPTER 3

Viewing and Printing Report Output

Topics:

- Displaying Reports in Hot Screen
- Scrolling a Report
- Displaying Reports in the Panel Facility
- Printing Reports
- Displaying Reports in the Terminal Operator Environment

Reports can be displayed on a terminal screen, sent to a printer, or routed to a file. FOCUS provides the Hot Screen facility and the Terminal Operator Environment for displaying reports on a screen, and the OFFLINE command and the Hot Screen facility for printing reports.

To display reports on a terminal screen, the value of the SET command PRINT parameter must be ONLINE, which is the default.

To send reports to a printer, the PRINT parameter must be set to OFFLINE.

This topic describes how to:

- Display reports in Hot Screen.
- Display reports in the Terminal Operator Environment.
- Display reports with the Panel facility.
- Preview a report without accessing the data.
- Send reports to a printer.
- Route reports to a file.

Displaying Reports in Hot Screen

By default, FOCUS reports are displayed in Hot Screen, the FOCUS full-screen output facility that enables you to scroll within a report, store report data in a separate file, and print a report.

This topic describes how to:

- Activate Hot Screen.
- Use PRINTPLUS.
- Control the Display of Emptying Reports.
- Access Help information.
- Scroll a report.
- Save selected data on reports.
- Locate character strings.
- Repeat commands.
- Redisplay reports.
- Display BY fields with panels.
- Scroll by columns of BY fields on panels.

Many of these functions can be invoked by using keys or by issuing commands at the command line. You can abbreviate a command name by using its shortest unique truncation.

Syntax

How to Activate Hot Screen

FOCUS automatically activates Hot Screen every time you start a FOCUS session.

To check if Hot Screen is activated, issue

```
? SET
```

at the FOCUS command line. The value of SCREEN should be ON.

If you are using a full-screen terminal, you can activate Hot Screen by issuing

```
SET SCREEN=ON
```

at the FOCUS command line. This is the default setting for full-screen terminals. Other acceptable values you can set SCREEN to are OFF and PAPER.

- If SCREEN is set to OFF, then Hot Screen is inactive. In this setting, FOCUS displays report output in line mode. It is the only setting for line terminals.
- If SCREEN is set to PAPER, Hot Screen is active and FOCUS uses the settings for the LINES and PAPER parameters to set the format of the screen display. The default settings are LINES=57 and PAPER=66. See the *Developing Applications* manual for more information about the LINES and PAPER parameters.

Use SET SCREEN=PAPER when you want the report display on your full-screen terminal to match the printed report.

Note: You can reset the SCREEN parameter with both the SET SCREEN command or the ON TABLE SET SCREEN command in a report request.

Using PRINTPLUS

PRINTPLUS includes enhancements to the display alternatives offered by the FOCUS Report Writer. For example, you might wish to place a FOOTING after a SUBFOOT in your report. PRINTPLUS provides the flexibility to produce the exact report you desire.

The PRINTPLUS parameter must be set to ON (the default) to use the TABLE capabilities. With PRINTPLUS on, the following occur:

- PAGE-BREAK is handled internally to provide the correct spacing of pages. For example, if a new report page is started and an instruction to skip a line at the top of the new page is encountered, FOCUS now knows to suppress the blank line and start at the top of the page.
- NOSPLIT is handled internally. (Use NOSPLIT to force a break at a specific spot.)
- You can perform RECAPs in cases where pre-specified conditions are met.
- A Report SUBFOOT now prints above the footing instead of below it.

Note:

- PRINTPLUS is not supported with StyleSheets. A warning message is generated in this case.
- Problems may be encountered if HOTSCREEN is set OFFLINE. A warning message is generated.

Syntax

How to Use PRINTPLUS

Issue the command

```
SET PRINTPLUS = {ON|OFF}
```

Example

Using PRINTPLUS With SUBFOOT and FOOTING

With PRINTPLUS on (the default), the SUBFOOT prints first, followed by the FOOTING.

```
USE CAR FOCUS F
END

TABLE FILE CAR
PRINT CAR MODEL
BY SEATS BY COUNTRY
IF COUNTRY EQ ENGLAND OR FRANCE OR ITALY
ON TABLE SUBFOOT
" "
" SUMMARY OF CARS IN COUNTRY BY SEATING CAPACITY"
FOOTING
" RELPMEK CAR SURVEY "
END
```

The output is:

SEATS	COUNTRY	CAR	MODEL
----	-----	---	-----
2	ENGLAND	TRIUMPH	TR7
	ITALY	ALFA ROMEO	2000 GT VELOCE
		ALFA ROMEO	2000 SPIDER VELOCE
		MASERATI	DORA 2 DOOR
4	ENGLAND	JAGUAR	V12XKE AUTO
		JENSEN	INTERCEPTOR III
	ITALY	ALFA ROMEO	2000 4 DOOR BERLINA
5	ENGLAND	JAGUAR	XJ12L AUTO
	FRANCE	PEUGEOT	504 4 DOOR

SUMMARY OF CARS IN COUNTRY BY SEATING CAPACITY
RELPMEK CAR SURVEY

Controlling the Display of Empty Reports

The command SET EMPTYREPORT enables you to control the output generated when a TABLE request retrieves zero records.

Issue this command from the command line

```
SET EMPTYREPORT = {ON|OFF}
```

where:

ON

Generates an empty report when zero records are found.

OFF

Does not generate a report when zero records are found. OFF is the default setting.

The command may also be issued from within a request. For example:

```
ON TABLE SET EMPTYREPORT ON
```

Note:

- TABLEF is not supported with SET EMPTYREPORT. When a TABLEF request retrieves zero records, EMPTYREPORT behaves as if EMPTYREPORT is set ON.
- This is a change in default behavior from prior releases of FOCUS. To restore prior default behavior, issue the SET EMPTYREPORT = ON command.
- SET EMPTYREPORT = OFF is not supported for HOLD FORMAT WP files.
- SET EMPTYREPORT = ON behaves as described above regardless of ONLINE or OFFLINE settings.

Accessing Help Information

To access help information about PF key assignments in Hot Screen, press PF1:

KEYS: 1=HELP 2=FENC 3=END 4=OFFL 5=LOCA 6=SAVE 7=BACK 8=FORW 10=LEFT 11=RIGHT

To view additional information about PF keys, press PF1 a second time.

To clear the Help window, press PF1 a third time.

You can also issue the SET HOTMENU command to display the Hot Screen PF key legend at the bottom of the Hot Screen report. For more information about the SET HOTMENU command, see the *Developing Applications* manual.

Scrolling a Report

You can use Hot Screen PF keys or commands to scroll within a report.

This section describes the keys and commands you use to scroll:

- Forward
- Backward
- Horizontally
- From fixed columns

Scrolling Forward

To scroll a report forward one page at a time, press PF8. Hot Screen displays the bottom two lines of the previous screen as the top two lines of the next screen.

When there are no more report lines, FOCUS displays the END-OF-REPORT message at the bottom of the screen. To clear this message and the end of the report, press ENTER. Hot Screen will return to the FOCUS command line.

You can also issue the following commands at the bottom of the screen to scroll forward through a report:

BOTTOM	Scrolls the display directly to the last page of the report.
NEXT <i>n</i>	Scrolls the display forward by the number of pages you specify.
FORW <i>n</i>	Like NEXT, scrolls the display forward the number of pages you specify.
DOWN <i>n</i>	Like NEXT and FORW, scrolls the display forward the number of pages you specify.

Note: If omitted, *n* defaults to 1.

Scrolling Backward

To scroll backward from the bottom of a report, press PF7.

You can also use the following commands to scroll backward through the report:

TOP	Scrolls the display directly back to the first page of the report.
UP <i>n</i>	Scrolls the display back the number of pages you specify.
BACK <i>n</i>	Like UP, scrolls the display back the number of pages you specify.

Note: If omitted, *n* defaults to 1.

Scrolling Horizontally

When a report exceeds the width of a screen, you can view it by scrolling horizontally to the left and to the right.

FOCUS displays the following symbol in the bottom right corner of the screen when the report is too wide:

`MORE =>`

You can also have Hot Screen scroll directly back to your first report screen.

- To scroll horizontally to the left one screen, press PF10. You can also issue:

`LEFT n`

where *n* is the number of characters. If *n* is omitted, it defaults to half of a screen.

- To scroll horizontally to the right one screen, press PF11 or issue:

`RIGHT n`

where *n* is the number of characters. If *n* is omitted, it defaults to 4 characters.

- To scroll directly back to the first screen, press PF9 or issue:

`RESET`

If you wish to scroll horizontally from a particular column, move the cursor to that location and press PF10 to scroll left or PF11 to scroll right.

Scrolling From Fixed Columns (Fencing)

To help you view a wide report in Hot Screen, you can hold the display of sort fields in the left-most columns of the screen while you scroll horizontally to the right to view the remaining columns.

To define a block of fixed columns, the steps are:

1. Scroll the display to the start of the first column to be held.
2. Press PF2.
3. Move the cursor to the end of the last column to be held.
4. Press PF2 again.

Scrolling Report Headings

You can make report headings and footers scroll along with the report contents in your HotScreen report by using the SET BYSCROLL command. Headings and footers scroll along with data to avoid confusion in matching the data with a corresponding header or footer.

To scroll report headings along with data the syntax is:

```
SET BYSCROLL = {ON|OFF}
```

where:

ON

Enables BYSCROLL.

OFF

Disables BYSCROLL. OFF is the default.

In order to use BYSCROLL, the text in the report must be longer than 80 characters and BYPANEL must be set ON. With BYPANEL OFF, headings and footings will not scroll. Note that fencing is not supported while BYPANEL is on. To determine the setting of BYSCROLL, enter ? SET ALL. (? SET does not display BYSCROLL.)

Saving Selected Data

Hot Screen also enables you to select and save data from a report request for use in subsequent requests. The steps are:

1. Position the cursor under the first character of the text to be saved.
2. Press PF6. FOCUS will save the text from that start character to the end of the line in a file with the file name SAVE. See Chapter 11, *Saving and Reusing Report Output*, for information about SAVE files.

Each time you repeat these steps, new text is appended to the SAVE file.

Locating Character Strings

To locate a character string in a report, the steps are:

1. Press PF5. FOCUS will prompt for the string:

`ENTER STRING TO LOCATE /`

2. Type the string you want to locate and press Enter.

FOCUS will search from the current position forward. When the string is located, the cursor is placed under the first occurrence of the string in the report. To locate additional instances of the string, press Enter for each instance. If the string is not found, a message is displayed at the bottom of the screen.

You can also issue the following command from the command line:

`LOCATE/string`

Repeating Commands

If you want to use a command repeatedly, issue it with a doubled first letter.

For example:

`RRIGHT 5`

After the command is executed, it will remain on the command line and can be repeated by pressing Enter.

You can cancel a command implicitly, by using a key command, or explicitly, by tabbing the cursor down to the command line and overwriting it with another command or spaces.

Redisplaying Reports

To redisplay reports immediately after you clear the last display, issue the command:

```
RETYPE
```

RETYPE only redisplay the report; the retrieval process is not repeated.

You can also use the RETYPE command to reformat specific fields in the report. The syntax is

```
RETYPE [field1/format1 ... fieldn/formatn]
```

where:

field1

Is a field name from the previous report request. It can be the full field name, alias, qualified field name, or unique truncation.

format1

Is the format of the field whose field type (D, I, P, F) is the same as the original field in the request. All formats except alpha (A), text (TX), dates, and fields with date edit options are supported.

When no arguments are provided, RETYPE redisplay the report. When one or more arguments are supplied, RETYPE redisplay the entire report and reformats the specified fields to the new format.

Note:

- RETYPE with a reformatted field does not recognize labels in EMR.
- When reformatting a packed field, the number of places after the decimal point may not be changed. For example, a P7 field can be redisplayed as P9 or P12.0C, but not as P9.2.
- You can save the internal matrix and issue a RETYPE later in the session if SAVEMATRIX is set ON (see the *Developing Applications* manual).
- You can issue any number of RETYPE commands one after the other:

```
TABLE FILE EMPLOYEE
```

```
.
```

```
.
```

```
.
```

```
END
```

```
RETYPE
```

```
RETYPE
```


Previewing Your Report

You can also preview the format of a report without actually accessing any data. The SET XRETRIEVAL command enables you to perform TABLE, TABLEF, or MATCH FILE requests and produce HOLD Master Files without processing the report. The syntax is

SET XRETRIEVAL = {OFF|ON}

where:

OFF

Specifies that no retrieval is to be performed.

ON

Specifies retrieval is to be performed. ON is the default.

SET XRETRIEVAL may also be issued from within a FOCUS request.

Displaying BY Fields With Panels

Hot Screen also enables you to display BY fields in the left portion of each panel of multi-panel reports. BY fields are vertical sort fields (see Chapter 4, *Sorting Tabular Reports*). The non-BY fields are displayed on the right portion of the panel. BY paneling is also available for OFFLINE reports.

To enable the display of BY fields with panels, set the BYPANEL parameter to one of the following values before issuing the request:

ON	Displays all BY fields specified in the report on each panel and prevents column splitting.
<i>n</i>	<p>Is the number of BY fields to be displayed; <i>n</i> is less than or equal to the total number of BY fields, specified in the request, from the major sort (first BY field) down. Column splitting is prevented.</p> <p>Column splitting occurs when a report column is too large to fit on the defined panel. By default, FOCUS splits the column, displaying as many characters as possible, and the remaining characters continue on the next panel.</p>
0	Zero displays BY fields on only the first panel. Column splitting is prevented.
OFF	Displays BY fields on the first panel only. Column splitting is permitted. This is the default.

In the following example, SET BYPANEL=ON displays the BY fields COUNTRY and CAR on each panel:

```
SET BYPANEL=ON
TABLE FILE CAR
PRINT SEG.LENGTH BY COUNTRY BY CAR
WHERE COUNTRY EQ 'ENGLAND'
END
```

PAGE 1.1

COUNTRY	CAR	LENGTH	WIDTH	HEIGHT	WEIGHT	WHEELBASE
ENGLAND	JAGUAR	190	66	48	3,435	105.0
		199	70	54	4,200	112.8
	JENSEN	188	69	53	4,000	105.0
	TRIUMPH	165	66	50	2,241	85.0

PAGE 1.2

COUNTRY	CAR	FUEL_CAP	BHP	RPM	MPG	ACCEL
ENGLAND	JAGUAR	18.0	241	5750	16	7
		24.0	241	5750	9	9
	JENSEN	24.0	385	4700	11	8
	TRIUMPH	14.5	90	5000	25	0

Reference

BYPANEL Conditions

- In Hot Screen, the panel width for the SET BYPANEL command is the physical screen width. The SET PANEL command will be ignored.
- In OFFLINE reports, the SET PANEL command is respected when used with SET BYPANEL. If you choose to override the report width, using the SET PANEL command, define a panel large enough to enable the BYPANEL feature. The panel size should accommodate all the BY fields in the request plus one non-BY field. If the defined panel is too small, the BYPANEL feature is disabled for the request and you receive a FOCUS error message.
- In OFFLINE reports, the SET BYPANEL command only works for widths of up to 132 characters.
- When SET BYPANEL is specified, the maximum number of panels is 99. When SET BYPANEL is OFF, the maximum number of panels is 4.
- BYPANEL may not be set from within a TABLE request using the ON TABLE SET command.
- Setting SCREEN=PAPER respects the SET BYPANEL command.
- The BYPANEL command may truncate summary text.

- The `BYPANEL = ON` command may truncate heading text. The heading will be repeated from the beginning on the following panels.
- `FOCUS` treats the `OVER` phrase as a physical block when it is used with the `BYPANEL` feature. As a result, `FOCUS` may split the column even though you have specified `BYPANEL`.
- In a request with several display commands, the number of `BY` fields in the first display command determines the `BY` field count for the `BYPANEL` command.
- You may not use the `FOLD-LINE` and `IN` to position columns with the `BYPANEL` command.
- You may not use `BYPANEL` with the `GRAPH` facility.

Scrolling by Columns of BY Fields in Panels

When a report is wider than the screen width and the `SET COLUMNS` command is specified, you can scroll columns using PF keys:

- To move to the right one column, press PF10.
- To move to the left one column, press PF11.
- To move up within the same column, press PF7.
- To move down within the same column, press PF8.

The SET COLUMNS Command

To enable column scrolling described in this section, specify the `SET COLUMNS` command as `ON`. To turn column scrolling off, specify `SET COLUMNS` as `OFF`.

Note the following usage information:

- If you specify the panel feature (`SET PANEL`), the panel size must be greater than the screen width in order for you to perform column scrolling.
- You cannot control column scrolling from within a `TABLE` request using the `ON TABLE SET` command.
- Report output must extend beyond the screen for column scrolling to have an effect.
- The `OVER` formatting option is not supported.
- Column width is determined by either the column title or field format, whichever is larger.
- When `COLUMNS` and `BYPANEL` are both set to `ON`, column scrolling is not enabled.
- Heading and footing lines are not maintained across the report as you scroll.

Displaying Reports in the Panel Facility

The Panel facility enables you to view reports that are too wide to fit on a typical 80-character terminal screen by dividing the display into a maximum of four panels. Pages are automatically numbered with decimal notation indicating the panel number (for example, 1.1, 1.2, 1.3), so that the results can be easily referenced. When these pages are produced as hardcopy, the page numbers also help you place the panels side by side. This feature is also very useful for reports over the 132-character standard line printer width.

To panel your report, issue

```
SET PANEL=n
```

before a report request. *n* is the number of characters you want displayed in each panel. This number must be in the range of 40 to 130.

For example:

```
SET PANEL=73
TABLE FILE EMPLOYEE
.
.
.
END
```

However, if you did not issue the panel command and the request has already been executed, FOCUS will automatically prompt you for a panel width:

```
REPORT WIDTH IS ### IT EXCEEDS TERMINAL PRINT LINE OF 130
TO PROCEED ENTER A PANEL WIDTH (40-130) OR 0 TO END =
```

At that point, you can either enter a number between 40 and 130, or enter 0 to end the report request.

Note:

- If the SET BYPANEL command is specified, the SET PANEL command is ignored for reports displayed in Hot Screen and the terminal screen can be divided into a maximum of 99 panels.
- The PANEL setting is ignored if StyleSheets are enabled.

Printing Reports

You can print reports by issuing a command or by pressing a function key while in Hot Screen.

The OFFLINE Command

You can use the OFFLINE command to send reports directly to a printer or a file without first displaying them on the screen. Simply issue the command

`OFFLINE`

before a report request.

Generally, this will direct all offline reports to the default output spool file. This file is assigned automatically when FOCUS is entered, and is in almost all cases a printer.

However, if you already issued the report request to be displayed online, you can still send its output to the printer by simply entering

`OFFLINE`

and then

`RETYPE`

at the FOCUS command line

You can also direct report output to a printer from within a report request by using the ON TABLE command:

`ON TABLE SET PRINT OFFLINE`

You can use OFFLINE to send reports to a file by allocating the ddname OFFLINE, as device type DISK, to the desired file using the FILEDEF command under CMS, or the FOCUS DYNAM command under MVS. The FILEDEF and DYNAM commands are described in the *Overview and Operating Environments* manual.

You can reroute report output to your screen by issuing

`ONLINE`

at the FOCUS command line, or reroute it only for the current request by including an ON TABLE SET PRINT ONLINE command in the request. Be sure to also issue the command

`OFFLINE CLOSE`

to close any current spool file and enable you to allocate new ones.

Printing Reports in Hot Screen

To send all or part of a report displayed in Hot Screen to an OFFLINE output device, press PF4.

The following print menu is displayed at the bottom of the screen:

1-Print entire report 2-Print this page 3-Cancel 4-Hold

1. Reformats the report to current page settings and then sends the entire report to a printer.
2. Prints the report page displayed, as formatted on the terminal screen.
3. Removes the print menu.
4. Creates a HOLD file using the entire report. The Master and FOCTEMP files will have the default name HOLD.

Press the desired number key (not function key) and then press Enter.

Displaying Reports in the Terminal Operator Environment

The FOCUS Terminal Operator Environment, discussed in the *Overview and Operating Environments* manual, provides a Table window that displays the report of the most recently executed report request. This enables you to view the report again without resubmitting the request. Unlike the RETYPE command, the most recent report is available even if other commands have been issued after the request.

The Table window displays a TABLE report as soon as you have terminated the report in Hot Screen. It holds up to the first 10 pages of report data. That is up to 200 lines that are up to a width of 130 characters.

Note: The Table Window does not record TABLEF reports, offline reports, or reports issued while the FOCUS SET SCREEN command is set to OFF.

CHAPTER 4

Sorting Tabular Reports

Topics:

- Sorting Rows
- Sorting Columns
- Sorting Rows and Columns
- Specifying the Sort Order
- Grouping Numeric Data Into Ranges
- Restricting Sort Field Values by Highest/Lowest Rank
- Aggregating and Sorting Report Columns
- Ranking Sort Field Values
- Hiding Sort Values
- Sorting With Multiple Display Commands
- Improving Efficiency With External Sorts

Sorting enables you to group or organize report information vertically and horizontally, in rows and columns, and specify a desired sequence.

The sort field organizes the rows and columns and controls the sequence of data items in the report. Any field in the data source can be the sort field. If you wish, you can select several sort fields, nesting one within another. Sort fields display when their values change.

You sort a report using BY and ACROSS phrases:

- BY displays the sort field values vertically, creating rows. Vertical sort fields are displayed in the left-most columns of the report.
- ACROSS displays the sort field values horizontally, creating columns. Horizontal sort fields are displayed across the top of the report.
- BY and ACROSS used in the same report create rows and columns, producing a grid or matrix.

Additional sorting options include the following:

- Sorting from low-to-high values or high-to-low values, and defining your own sorting sequence.
- Grouping numeric data into ranges.
- Ranking data and selecting data by rank.
- Leaving the sort field's value out of the report.
- Aggregating and sorting numeric columns simultaneously.

Reference

Sorting and Displaying Data

There are two ways that you can sort information, depending on the type of display command you use:

- You can sort and display individual values of a field using the PRINT or LIST command.
- You can group and aggregate information, for example, showing the number of field occurrences per sort value using the COUNT command, or summing the field values using the SUM command.

When you use the display commands PRINT and LIST, the report may generate several rows per sort value—specifically, one row for each occurrence of the display field. When you use the commands SUM and COUNT, the report generates one row for each unique set of sort values. For related information, see *Sorting With Multiple Display Commands* on page 4-31.

For details on all display commands, see Chapter 2, *Displaying Report Data*.

Sorting Rows

You can sort report information vertically using the BY phrase. This creates rows in your report.

You can include up to 32 BY phrases per report request (31 if using PRINT or LIST display commands).

Syntax

How to Sort by Rows

The syntax for the BY phrase is

`BY sortfield`

where:

`sortfield`

Is the name of the sort field.

Reference

Usage Notes for Sorting Rows

- When using the display command LIST with a BY phrase, the LIST counter is reset to 1 each time the major sort value changes.
- Each sort field value appears only once in the report. For example, if there are six employees in the MIS department, a request that declares
`PRINT LAST_NAME BY DEPARTMENT`
prints MIS once, followed by six employee names.
- The default sort sequence is low-to-high, with the following variations for different operating systems. In MVS and VM the sequence is a-z, A-Z, 0-9 for alphanumeric fields; 0-9 for numeric fields. In UNIX and NT the sequence is 0-9, A-Z, a-z for alphanumeric fields; 0-9 for numeric. You can specify other sorting sequences, as described in *Specifying the Sort Order* on page 4-10.
- You cannot use text fields as sort fields (text fields are those described in the Master File with a FORMAT value of TX).
- You cannot use a temporary field created by a COMPUTE command as a sort field. However, you can accomplish this indirectly by first creating a HOLD file that includes the field and then reporting from the HOLD file (HOLD files are described in Chapter 11, *Saving and Reusing Your Report Output*). However, you can use a temporary field created by a DEFINE command, or by the DEFINE attribute in a Master File, as a sort field.
- If you specify several sort fields when reporting from a multi-path data source, all the sort fields must be in the same path.
- Sort phrases cannot contain format information for fields.

Example **Sorting Rows With BY**

To display all employee IDs for each department, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID
BY DEPARTMENT
END
```

The output displays a row for each EMP_ID in each department:

PAGE 1

DEPARTMENT	EMP_ID
MIS	112847612
	117593129
	219984371
	326179357
	543729165
	818692173
PRODUCTION	071382660
	119265415
	119329144
	123764317
	126724188
	451123478

Using Multiple Sort Fields With BY

You can organize information in a report using more than one sort field. When you specify several sort fields, the sequence of the BY phrases determines the sort order: the first BY phrase sets the major sort break, the second BY phrase sets the second sort break, and so on. Each successive sort is nested within the previous one.

Example**Sorting With Multiple BYs**

The following request uses multiple BYs to sort rows:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY DEPARTMENT BY LAST_NAME
WHERE CURR_SAL GT 21500
END
```

The output is:

PAGE 1

DEPARTMENT	LAST_NAME	CURR_SAL
MIS	BLACKWOOD	\$21,780.00
	CROSS	\$27,062.00
PRODUCTION	BANNING	\$29,700.00
	IRVING	\$26,862.00

Sorting Columns

You can sort report information horizontally using the ACROSS phrase. This creates columns in your report. You can have up to five ACROSS phrases per report request. Each ACROSS phrase can retrieve up to 95 sort field values. The total number of ACROSS columns is equal to the total number of ACROSS sort field values multiplied by the total number of display fields. The maximum number of display fields your report can contain is determined by a combination of factors. For details, see Chapter 1, *Creating Tabular Reports*.

You can also produce column totals for ACROSS sort field values using ACROSS-TOTAL. For details see, *Producing Column Totals With ACROSS-TOTAL* on page 4-7.

Syntax**How to Sort Columns**

The syntax for the ACROSS phrase is

```
ACROSS sortfield
```

where:

```
sortfield
```

Is the name of the sort field.

Reference

Usage Notes for Sorting Columns

- Each sort field value is displayed only once in the report. For example, if there are six employees in the MIS department, a report that declares
`PRINT LAST_NAME ACROSS DEPARTMENT`
will print MIS once, followed by six employee names.
- You cannot use text fields as sort fields (text fields are those described in the Master File with a FORMAT value of TX).
- You cannot use a temporary field created by a COMPUTE command as a sort field. However, you can accomplish this indirectly by first creating a HOLD file that includes the field and then reporting from the HOLD file (HOLD files are described in Chapter 11, *Saving and Reusing Your Report Output*). You can use a temporary field created by a DEFINE command, or by the DEFINE attribute in a Master File, as a sort field.
- For an ACROSS phrase, the SET SPACES command controls the distance between ACROSS sets. For more information, see Chapter 9, *Customizing Tabular Reports*.
- Sort phrases cannot contain format information for fields.
- If you specify several sort fields when reporting from a multi-path data source, all the sort fields must be in the same path.

Example

Sorting Columns With ACROSS

To show each department’s salary outlay, issue the following request:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL ACROSS DEPARTMENT
END
```

The output is:

PAGE 1	
DEPARTMENT	
MIS	PRODUCTION

\$108,002.00	\$114,282.00

Notice that the horizontal sort displays a column for each sort field (department).

Using Multiple Sort Fields With ACROSS

You can sort a report using more than one sort field. When several sort fields are used, the ACROSS phrase order determines the sorting order: the first ACROSS phrase sets the first sort break, the second ACROSS phrase sets the second sort break, and so on. Each successive sort is nested within the previous one.

Example

Sorting With Multiple ACROSS Phrases

The request sorts the sum of current salary, first by department, then by job code.

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
ACROSS DEPARTMENT ACROSS CURR_JOBCODE
WHERE CURR_SAL GT 21500
END
```

The output is:

PAGE 1			
DEPARTMENT			
MIS		PRODUCTION	
CURR_JOBCODE			
A17		B04	
		A15	
		A17	
\$27,062.00		\$21,700.00	
		\$26,062.00	
		\$29,700.00	

Producing Column Totals With ACROSS-TOTAL

ACROSS-TOTAL produces totals for columns of numbers created by an ACROSS sort phrase. The display of data on a report makes the report simple to read and understand. Integer, single precision floating point, double precision floating point, packed, and long packed fields can all be totaled.

ACROSS-TOTAL differs from ROW-TOTAL in that ACROSS-TOTAL only totals the ACROSS column data, excluding the sorted column data that ROW-TOTAL displays.

Syntax

How to Produce Column Totals With ACROSS-TOTAL

```
ACROSS sortfield ACROSS-TOTAL [AS 'name'] [COLUMNS col1 AND col2 ...]
```

where:

```
sortfield
```

Is the name of the field being sorted across.

```
name
```

Is the new name for the ACROSS-TOTAL column title.

```
col1, col2
```

Are the titles of the ACROSS columns you want to include in the total.

Example

Requesting ACROSS-TOTAL in a Report

```
TABLE FILE MOVIES
SUM COPIES BY CATEGORY
COUNT TITLE BY CATEGORY
ACROSS RATING ACROSS-TOTAL
COLUMNS PG AND R
END
```

The output is:

CATEGORY	COPIES	RATING		
		PG	R	TOTAL
		TITLE	TITLE	TITLE
		COUNT	COUNT	COUNT

ACTION	14	2	3	5
COMEDY	16	4	1	5
DRAMA	2	0	1	1
FOREIGN	5	2	3	5
MUSICALS	2	1	1	2
MYSTERY	17	2	5	7
SCI/FI	3	0	3	3

Reference

Usage Notes for ACROSS-TOTAL

- Stacking headings in ACROSS-TOTAL is not allowed.
- Attempts to use ACROSS-TOTAL with other types of fields (alphanumeric, text, and dates) produces blank columns.
- In cases of multiple ACROSS columns with ACROSS-TOTAL there will be additional columns with subtotaled values.
- The results of ROW-TOTAL and ACROSS-TOTAL are the same if there is only a single display field or single display command in the procedure.
- The maximum number of ACROSS-TOTAL phrases is 5.

Sorting Rows and Columns

You can create a matrix report by sorting both rows and columns. When you include both BY and ACROSS phrases in a report request, information is sorted vertically and horizontally, turning the report into a matrix of information that you read like a grid. A matrix report can have several BY and ACROSS sort fields.

Example Creating a Simple Matrix

The following request displays total salary outlay across departments and by job codes, creating a matrix report.

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
ACROSS DEPARTMENT
BY CURR_JOBCODE
END
```

The output is:

```
PAGE      1
```

	DEPARTMENT	
	MIS	PRODUCTION
CURR_JOBCODE		
A01	.	\$9,500.00
A07	\$9,000.00	\$11,000.00
A15	.	\$26,862.00
A17	\$27,062.00	\$29,700.00
B02	\$18,480.00	\$16,100.00
B03	\$18,480.00	.
B04	\$21,700.00	\$21,120.00
B14	\$13,200.00	.

Example Creating a Matrix With Several Sort Fields

The following request uses several BY and ACROSS sort fields to create a matrix report.

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
ACROSS DEPARTMENT ACROSS LAST_NAME
BY CURR_JOBCODE BY ED_HRS
WHERE DEPARTMENT EQ 'MIS'
WHERE CURR_SAL GT 21500
END
```

The output is:

PAGE 1

		DEPARTMENT MIS LAST_NAME BLACKWOOD		CROSS
CURR_JOBCODE	ED_HRS			
A17	45.00			\$27,062.00
B04	75.00	\$21,780.00		.

Specifying the Sort Order

Sort field values are automatically displayed in ascending order, beginning with the lowest value and continuing to the highest value. The default sorting sequence varies for operating systems. On MVS and VM it is a-z, A-Z, 0-9 for alphanumeric fields; 0-9 for numeric fields. On UNIX and NT it is 0-9, A-Z, a-z for alphanumeric fields; 0-9 for numeric fields.

You have the option of overriding this default and displaying values in descending order, ranging from the highest value to the lowest value, by including HIGHEST in the sort phrase.

Syntax

How to Specify the Sort Order

```
{BY|ACROSS} {LOWEST|HIGHEST} sortfield
```

where:

LOWEST

Sorts in ascending order, beginning with the lowest value and continuing to the highest value (a-z, A-Z, 0-9 for alphanumeric fields; 0-9 for numeric fields). This option is the default.

HIGHEST

Sorts in descending order, beginning with the highest value and continuing to the lowest value. You can also use TOP as a synonym for HIGHEST.

sortfield

Is the name of the sort field.

Example

Sorting in Ascending Order

The following report request does not specify a particular sorting order, and so, by default, it lists salaries ranging from the lowest to the highest.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
BY CURR_SAL
END
```

You can explicitly specify this same ascending order by including LOWEST in the sort phrase.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
BY LOWEST CURR_SAL
END
```

The output is:

```
PAGE      1

      CURR_SAL  LAST_NAME
-----
$9,000.00 GREENSPAN
$9,500.00  SMITH
$11,000.00 STEVENS
$13,200.00  SMITH
$16,100.00 MCKNIGHT
$18,400.00  JONES
          MCCOY
$21,120.00  ROMANS
$21,700.00 BLACKWOOD
$26,862.00  IRVING
$27,062.00  CROSS
$29,700.00  BANNING
```

Example **Sorting in Descending Order**

The following request lists salaries ranging from the highest to lowest.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
BY HIGHEST CURR_SAL
END
```

The output is:

CURR_SAL	LAST_NAME
\$29,700.00	BANNING
\$27,062.00	CROSS
\$26,062.00	IRVING
\$21,780.00	BLACKWOOD
\$21,120.00	ROMANS
\$18,480.00	JONES
	MCCOY
\$16,100.00	MCKNIGHT
\$13,200.00	SMITH
\$11,000.00	STEVENS
\$9,500.00	SMITH
\$9,000.00	GREENSPAN

Specifying Your Own Sort Order

Sort field values are automatically displayed in ascending order, beginning with the lowest value and continuing to the highest value.

You can override the default order and display values in your own user-defined sorting sequence. To do this, you need to decide the following:

1. Which sort field values you want to allow. You can specify every sort field value or a subset of values. When you issue your report request, only records containing these values are included in the report.
2. In what order you want the values to appear. You can specify any order—for example, you could specify that an A1 sort field containing a single-letter code be sorted in the order A, Z, B, C, Y...

There are two ways to specify your own sorting order, depending on whether you are sorting rows with BY or sorting columns with ACROSS:

- BY field ROWS OVER for defining your own row sort sequence.
- ACROSS field COLUMNS AND for defining your own column sort sequence.

Syntax

How to Define Your Own Sort Order

```
BY sortfield ROWS value1 OVER value2 [... OVER valuen]
```

where:

sortfield

Is the name of the sort field.

value1

Is the sort field value that is first in the sorting sequence.

value2

Is the sort field value that is second in the sorting sequence.

valuen

Is the sort field value that is last in the sorting sequence.

An alternative syntax is

```
FOR sortfield value1 OVER value2 [... OVER valuen]
```

which uses the row-based reporting phrase FOR, described in Chapter 16, *Creating Financial Reports*.

Reference

Usage Notes for Defining Your Sort Order

- Any sort field value that you do not specify in the BY ROWS OVER phrase is not included in the sorting sequence and does not appear in the report.
- Sort field values that contain embedded blank spaces should be enclosed in single quotation marks.
- Any sort field value that you do specify in the BY ROWS OVER phrase is included in the report, whether or not there is data.
- The name of the sort field is not included in the report.
- Each report request can contain no more than one BY ROWS OVER phrase.

Example **Defining Your Row Sort Order**

To sort employees by the banks at which their paychecks are automatically deposited, and to define your own label in the sorting sequence for the bank field, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
BY BANK_NAME ROWS 'BEST BANK' OVER STATE
OVER ASSOCIATED OVER 'BANK ASSOCIATION'
END
```

The output is:

PAGE	1
	LAST_NAME

BEST BANK	BANNING
STATE	JONES
'ASSOCIATED	IRVING
ASSOCIATED	BLACKWOOD
ASSOCIATED	MCKNIGHT
BANK ASSOCIATION	CROSS

Syntax **How to Define Column Sort Sequence**

```
ACROSS sortfield COLUMNS value1 AND value2 [... AND valuen]
```

where:

sortfield

Is the name of the sort field.

value1

Is the sort field value that is first in the sorting sequence.

value2

Is the sort field value that is second in the sorting sequence.

valuen

Is the sort field value that is last in the sorting sequence.

Reference**Usage Notes for Defining Column Sort Sequence**

- Any sort field value that you do not specify in the ACROSS COLUMNS AND phrase is not included in the label within the sorting sequence and does not appear in the report.
- Sort field values that contain embedded blank spaces should be enclosed in single quotation marks.
- Any sort field value that you do specify in the ACROSS COLUMNS AND phrase is included in the report, whether or not there is data.

Example**Defining Column Sort Sequence**

To sum employee salaries by the bank at which they are automatically deposited, and to define your own label within the sorting sequence for the bank field, issue the following request:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
ACROSS BANK_NAME COLUMNS 'BEST BANK' AND STATE
      AND ASSOCIATED AND 'BANK ASSOCIATION'
END
```

The output is:

PAGE 1			
BANK_NAME BEST BANK	STATE	ASSOCIATED	BANK ASSOCIATION
\$29,790.00	\$18,480.00	\$64,742.00	\$27,062.00

Grouping Numeric Data Into Ranges

When you sort a report using a numeric sort field, you can group the sort field values together and define the range of each group.

There are two ways of defining groups:

- Defining groups of equal range using the IN-GROUPS-OF phrase.
- Defining groups of unequal range using the FOR phrase.

The FOR phrase is usually used to produce matrix reports and is part of the Financial Modeling Language (FML). However, you can also use it to create columnar reports that group sort field values in unequal ranges.

The FOR phrase displays the sort value for each individual row. The ranges do not have to be contiguous—that is, you can define your ranges with gaps between them.

The FOR phrase is described in more detail in Chapter 16, *Creating Financial Reports*.

Syntax

How to Define Groups of Equal Range

```
{BY|ACROSS} sortfield IN-GROUPS-OF value [TOP limit]
```

where:

sortfield

Is the name of the sort field. The sort field must be numeric—that is, its format must be I (integer), F (floating-point number), D (decimal number), or P (packed number).

value

Is the range by which sort field values will be grouped.

limit

Is an optional number that defines the highest group to be included in the report. This is a way of limiting the number of rows or columns in the report.

Reference

Usage Notes for Defining Groups of Equal Range

There are several usage considerations:

- The first sort field range starts from the lowest value.
- Each report request can contain up to five IN-GROUPS-OF phrases.
- The IN-GROUPS-OF phrase can only be used once per BY field.
- The value displayed is the end point of each range.

Example

Defining Groups of Equal Ranges

To show which employees fall into which salary ranges, and to define the ranges by \$5,000 increments, issue the following report request:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
BY CURR_SAL IN-GROUPS-OF 5000
END
```

The output is:

PAGE 1

CURR_SAL	LAST_NAME
-----	-----
\$5,000.00	SMITH
	GREENSPAN
\$10,000.00	STEVENS
	SMITH
\$15,000.00	JONES
	MCCOY
	MCKNIGHT
\$20,000.00	ROMANS
	BLACKWOOD
\$25,000.00	BANNING
	IRVING
	CROSS

Syntax

How to Define Custom Groups of Data Values

The syntax for using the FOR phrase to sort in unequal ranges is

```
FOR sortfield begin1 TO end1 [OVER begin2 TO end2 ... ]
```

where:

sortfield

Is the name of the sort field.

begin

Is a value that identifies the beginning of a range.

end

Is a value that identifies the end of a range.

Example

Defining Custom Groups of Data Values

The following request displays employee salaries, but it groups them in an arbitrary way. Notice that the starting value of each range prints in the report.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
FOR CURR_SAL
9000 TO 13500 OVER
14000 TO 19700 OVER
19800 TO 30000
END
```

The output is:

```
PAGE      1

      LAST_NAME
-----
9000  STEVENS
9000  SMITH
9000  SMITH
9000  GREENSPAN
14000 JONES
14000 MCCOY
14000 MCKNIGHT
19800 BANNING
19800 IRVING
19800 ROMANS
19800 BLACKWOOD
19800 CROSS
```


Grouping Numeric Data Into Tiles

You can group numeric data into any number of tiles (for example, percentiles or quartiles). This enables you to answer such questions as which salesmen are in the top half of all salesmen based on total sales, or which students are in the top ten percent (decile) based on test scores.

Grouping data in tiles sorts data instances on a BY field in the request and then apportions them as equally as possible into the number of tile groups you specify. The following occurs when you group data into tiles:

- A new column (labeled **TILE** by default) is added to the report output and displays the tile number assigned to each instance of the tile field. You can change the column heading with an **AS** phrase.
- Tiling is calculated within all of the higher-level sort fields in the request and restarts whenever a sort field at a higher level than the tile field's value changes.
- Instances are counted using the tile field. If the request prints fields from lower level segments, there may be multiple report lines that correspond to one instance of the tile field.
- Instances with the same tile field value are placed in the same tile. For example, consider the following data, which is to be apportioned into three tiles:

1
5
5
5
8
9

In this case, dividing the instances equally produces the following:

Group	Data Values
1	1,5
2	5,5
3	8,9

However, because all of the same data values must be in the same tile, the fives (5) that are in group 2 are moved to group 1. Group 2 remains empty. The final tiles are:

Tile Number	Data Values
1	1,5,5,5
2	
3	8,9

Syntax

How to Group Numeric Data Into Tiles

```
BY [ {HIGHEST|LOWEST} [k] ] tilefield [AS 'head1']  
    IN-GROUPS-OF n TILES [TOP m] [AS 'head2']
```

where:

HIGHEST

Sorts the data in descending order so that the highest data values are placed in tile 1.

LOWEST

Sorts the data in ascending order so that the lowest data values are placed in tile 1.
This is the default sort order.

k

Is a positive integer representing the number of tile groups to display in the report.
For example, BY HIGHEST 2 displays the two non-empty tiles with the highest data values.

tilefield

Is the field whose values are used to assign the tile numbers.

head1

Is a heading for the column that displays the values of the tile sort field.

n

Is a positive integer not greater than 32,767 specifying the number of tiles to be used in grouping the data. For example, 100 tiles produces percentiles, 10 tiles produces deciles.

m

Is a positive integer indicating the highest tile value to display in the report. For example, TOP 3 does not display any data row that is assigned a tile number greater than 3.

head2

Is a new heading for the column that displays the tile numbers.

Note:

- The syntax accepts numbers that are not integers for *k*, *n*, and *m*. On MVS and VM, values with decimals are rounded to integers; on UNIX and Windows NT they are truncated. If the numbers supplied are negative or zero, an error message is generated.
- Both *k* and *m* limit the number of rows displayed within each sort break in the report. If you specify both, the more restrictive value will control the display. If *k* and *m* are both greater than *n* (the number of tiles), *n* will be used.

Example**Grouping Data Into Five Tiles**

```
TABLE FILE MOVIES
PRINT TITLE
BY CATEGORY
BY LISTPR IN-GROUPS-OF 5 TILES
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
END
```

The output is:

CATEGORY	LISTPR	TILE	TITLE
-----	-----	----	-----
ACTION	14.95	1	TOP GUN
	19.95	2	JAWS
			RAMBO III
	19.98	4	ROBOCOP
	19.99	5	TOTAL RECALL
CHILDREN	14.95	1	SESAME STREET-BEDTIME STORIES AND SONGS
	14.98	1	ROMPER ROOM-ASK MISS MOLLY
	19.95	2	SMURFS, THE
			SCOOPY-DOO-A DOG IN THE RUFF
	26.99	3	BAMBI
	29.95	4	ALICE IN WONDERLAND
			SLEEPING BEAUTY
	44.95	5	SHAGGY DOG, THE

Note that the tiles are assigned within the higher-level sort field CATEGORY. The ACTION category does not have any data assigned to tile 3. The CHILDREN category has all five tiles.

Example **Displaying the First Three Tile Groups**

The following request prints only the first three tiles in each category:

```
TABLE FILE MOVIES
PRINT TITLE
BY CATEGORY
BY LOWEST 3 LISTPR IN-GROUPS-OF 5 TILES
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
END
```

The output is:

CATEGORY	LISTPR	TILE	TITLE
-----	-----	----	-----
ACTION	14.95	1	TOP GUN
	19.95	2	JAWS
			RAMBO III
CHILDREN	19.98	4	ROBOCOP
	14.95	1	SESAME STREET-BEDTIME STORIES AND SONGS
	14.98	1	ROMPER ROOM-ASK MISS MOLLY
	19.95	2	SMURFS, THE
			SCOOBY-DOO-A DOG IN THE RUFF
	26.99	3	BAMBI

Note that the request displays three tile groups in each category. Because no data was assigned to tile 3 in the ACTION category, tiles 1, 2, and 4 display for that category.

Example **Displaying Tiles With a Value of Three or Less**

In the following request, the TOP 3 phrase restricts the display to tile numbers less than or equal to 3:

```
TABLE FILE MOVIES
PRINT TITLE
BY CATEGORY
BY LOWEST 3 LISTPR IN-GROUPS-OF 5 TILES TOP 3
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
END
```

The output is:

CATEGORY	LISTPR	TILE	TITLE
-----	-----	----	-----
ACTION	14.95	1	TOP GUN
	19.95	2	JAWS
			RAMBO III
CHILDREN	14.95	1	SESAME STREET-BEDTIME STORIES AND SONGS
	14.98	1	ROMPER ROOM-ASK MISS MOLLY
	19.95	2	SMURFS, THE
			SCOOBY-DOO-A DOG IN THE RUFF
	26.99	3	BAMBI

Because no data was assigned to tile 3 in the ACTION category, only tiles 1 and 2 display for that category.

Example

Grouping Data Into Tiles and Customizing Column Headings

The following request changes the column headings for both the LISTPR and TILE columns:

```
TABLE FILE MOVIES
PRINT TITLE
BY CATEGORY
BY LISTPR AS 'PRICE' IN-GROUPS-OF 10 TILES TOP 3 AS 'DECILE'
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
END
```

The output is:

CATEGORY	PRICE	DECILE	TITLE
-----	-----	-----	-----
ACTION	14.95	1	TOP GUN
	19.95	3	JAWS
			RAMBO III
CHILDREN	14.95	1	SESAME STREET-BEDTIME STORIES AND SONGS
	14.98	2	ROMPER ROOM-ASK MISS MOLLY
	19.95	3	SMURFS, THE
			SCOOBY-DOO-A DOG IN THE RUFF

Reference

Usage Notes for Tiles

- If a request retrieves data from segments that are descendents of the segment containing the tile field, multiple report rows may correspond to one instance of the tile field. These additional report rows do not affect the number of instances used to assign the tile values. However, if you retrieve fields from multiple segments and create a single-segment extract file, this flat file will have multiple instances of the tile field, and this increased number of instances may affect the tile values assigned. Therefore, when you can run the same request against the multi-level file and the single-segment file, different tile assignments may result.
- Tiles are always calculated on a BY sort field in the request.
- Only one tiles calculation is supported per request. However, the request can contain up to five (the maximum allowed) non-tile IN-GROUP-OF phrases in addition to the TILES phrase.
- Comparisons for the purpose of assigning tile numbers use exact data values regardless of their display format. Therefore, if you display a floating-point value as D7, you may not be showing enough significant digits to indicate why values are placed in separate tiles.
- The tile field can be a real field or a virtual field created with a DEFINE command or a DEFINE in the Master File. The COMPUTE command cannot be used to create a tile field.

- Empty tiles do not display in the report output.
- In requests with multiple display commands, tiles are supported only at the lowest level and only with the BY LOWEST phrase.
- Tiles are supported with extract files. However, the field used to calculate the tiles will propagate three fields to a HOLD file unless you set HOLDLIST to PRINTONLY.
- Tiles are not supported with BY TOTAL, TABLEF, FML, and GRAPH.

Restricting Sort Field Values by Highest/Lowest Rank

When you sort report rows using the BY phrase, you can restrict the sort field values to a group of high or low values.

Syntax

How to Restrict Sort Field Values By Highest/Lowest Rank

`BY {HIGHEST n|LOWEST n} sortfield`

where:

`HIGHEST n`

Specifies that only the highest *n* sort field values will be included in the report. TOP is a synonym for HIGHEST.

`LOWEST n`

Specifies that only the lowest *n* sort field values will be included in the report.

`sortfield`

Is the name of the sort field. The sort field can be numeric or alphanumeric.

Reference

Usage Notes for Restricting Sort Fields

- HIGHEST/LOWEST *n* refers to the number of sort field values, not the number of report rows. If several records have the same sort field value that satisfies the HIGHEST/LOWEST *n* criteria, all of them are included in the report.
- You can have up to five sort fields with BY HIGHEST or BY LOWEST.

Example**Restricting Sort Field Values to a Group**

The following request displays the names of the employees earning the five highest salaries:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
BY HIGHEST 5 CURR_SAL
END
```

The output is:

PAGE 1

<u>CURR_SAL</u>	<u>LAST_NAME</u>
\$29,700.00	BANNING
\$27,062.00	CROSS
\$26,862.00	IRVING
\$21,780.00	BLACKWOOD
\$21,120.00	ROMANS

Aggregating and Sorting Report Columns

By including the phrase **BY TOTAL** in a request, you can apply aggregation and sorting simultaneously to numeric columns in your report in one pass of the data.

In addition, you can use the **BY TOTAL** phrase to sort based on temporary values that are calculated by the **COMPUTE** command. For details, see Chapter 6, *Creating Temporary Fields*.

For **BY TOTAL** to work correctly, you must have an aggregating display command such as **SUM**. A non-aggregating display command, such as **PRINT**, simply retrieves the data without aggregating it.

The records will be sorted in either ascending or descending sequence based on your query. Ascending order is the default.

Note: On MVS and VM, the sort on the aggregated value is calculated using an external sort package, even if **EXTSORT = OFF**.

Syntax

How to Sort and Aggregate a Report Column

```
[RANKED] BY [HIGHEST|LOWEST [n] ] TOTAL display_field
```

where:

RANKED

Adds a column to the report in which a rank number is assigned to each aggregated sort value in the report output. If multiple rows have the same ranking, the rank number only appears in the first row.

n

Is the number of sort field values you wish to display in the report. If *n* is omitted, all values of the calculated sort field are displayed. Low to high order is the default.

display_field

Can be a field name, a field name preceded by an operator (that is, prefixoperator.fieldname), or a calculated value.

A BY TOTAL field is treated as a display field when the internal matrix is created. After the matrix is created, the output lines are aggregated and re-sorted based on all of the sort fields.

Example

Sorting and Aggregating by Report Columns

In this example, the average of the wholesale prices is calculated and used as a sort field, and the highest two are displayed:

```
TABLE FILE MOVIES
SUM  WHOLESALEPR CNT.WHOLESALEPR
BY CATEGORY
   BY HIGHEST 2 TOTAL AVE.WHOLESALEPR AS 'AVE.WHOLESALEPR'
   BY RATING
WHERE CATEGORY EQ 'CLASSIC' OR 'FOREIGN' OR 'MUSICALS'
END
```

The output is:

CATEGORY	AVE.WHOLESALEPR	RATING	WHOLESALEPR	WHOLESALEPR COUNT
CLASSIC	40.99	G	40.99	1
	16.08	NR	160.80	10
FOREIGN	31.00	PG	62.00	2
	23.66	R	70.99	3
MUSICALS	15.00	G	15.00	1
	13.99	PG	13.99	1
		R	13.99	1

Example Aggregating and Ranking Sort Field Values

In this example, the average of the wholesale prices is calculated and used as a sort field, the highest two are displayed and ranked.

```
TABLE FILE MOVIES
SUM WHOLESALEPR CNT.WHOLESALEPR
BY CATEGORY
RANKED BY HIGHEST 2 TOTAL AVE.WHOLESALEPR AS 'AVE.WHOLESALEPR'
BY RATING
WHERE CATEGORY EQ 'CLASSIC' OR 'FOREIGN' OR 'MUSICALS'
END
```

The output is:

CATEGORY	RANK	AVE.WHOLESALEPR	RATING	WHOLESALEPR	WHOLESALEPR COUNT
CLASSIC	1	40.99	G	40.99	1
	2	16.08	NR	160.80	10
FOREIGN	1	31.00	PG	62.00	2
	2	23.66	R	70.99	3
MUSICALS	1	15.00	G	15.00	1
	2	13.99	PG	13.99	1
			R	13.99	1

Ranking Sort Field Values

When you sort report rows using the BY phrase, you can indicate the numeric rank of each row. Ranking sort field values is frequently combined with restricting sort field values by rank.

You can rank aggregated values using the syntax RANKED BY TOTAL. For details, see *How to Sort and Aggregate a Report Column* on page 4-26.

Syntax How to Rank Sort Field Values

```
RANKED BY sortfield
```

where:

```
sortfield
```

Is the name of the sort field. The field can be numeric or alphanumeric.

Reference Usage Notes for Ranking Sort Field Values

- You can replace the RANK column heading with one that you define by using the AS phrase. For more information, see Chapter 9, *Customizing Tabular Reports*.
- Several report rows will have the same rank if they have identical sort field values.

Example **Ranking Sort Field Values**

Issue the following request to display a list of employee names in salary order, indicating the rank of each employee by salary:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
RANKED BY CURR_SAL
END
```

The output is:

RANK	CURR_SAL	LAST_NAME
1	\$9,000.00	GREENSPAN
2	\$9,500.00	SMITH
3	\$11,000.00	STEVENS
4	\$13,200.00	SMITH
5	\$16,100.00	MCKNIGHT
6	\$18,480.00	JONES
		MCCOY
7	\$21,120.00	ROMANS
8	\$21,780.00	BLACKWOOD
9	\$26,862.00	IRVING
10	\$27,062.00	CROSS
11	\$29,700.00	BANNING

Example **Ranking and Restricting Sort Field Values**

Ranking sort field values is frequently combined with restricting sort field values by rank, as in the following example:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
RANKED BY HIGHEST 5 CURR_SAL
END
```

The output is:

RANK	CURR_SAL	LAST_NAME
1	\$29,700.00	BANNING
2	\$27,062.00	CROSS
3	\$26,862.00	IRVING
4	\$21,780.00	BLACKWOOD
5	\$21,120.00	ROMANS

Hiding Sort Values

When you sort a report, you can omit the sort field value itself from the report using the phrase NOPRINT. This can be helpful in several situations, for instance, when you use the same field as a sort field and a display field.

Syntax

How to Hide Sort Values

```
{BY|ACROSS} sortfield {NOPRINT|SUP-PRINT}
```

where:

sortfield

Is the name of the sort field.

You can use SUP-PRINT as a synonym for NOPRINT.

Example

Hiding Sort Values

If you want to display a list of employee names sorted in alphabetical order, the following request is insufficient.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME
END
```

The output lists the names in the order that they were entered into the data source:

```
PAGE      1

LAST_NAME      FIRST_NAME
-----
STEVENS        ALFRED
SMITH          MARY
JONES          DIANE
SMITH          RICHARD
BANNING        JOHN
IRVING         JOAN
ROMANS         ANTHONY
MCCOY          JOHN
BLACKWOOD      ROSEMARIE
MCKNIGHT       ROGER
GREENSPAN      MARY
CROSS          BARBARA
```

To list the employee names in alphabetical order, you would sort the LAST_NAME field by the LAST_NAME field and hide the sort field occurrence using the phrase NOPRINT.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME
BY LAST_NAME NOPRINT
END
```

This request generates the desired output:

PAGE 1

LAST_NAME	FIRST_NAME
-----	-----
BANNING	JOHN
BLACKWOOD	ROSEMARIE
CROSS	BARBARA
GREENSPAN	MARY
IRVING	JOAN
JONES	DIANE
MCCOY	JOHN
MCKNIGHT	ROGER
ROMANS	ANTHONY
SMITH	MARY
SMITH	RICHARD
STEVENS	ALFRED

Sorting With Multiple Display Commands

A request can consist of up to sixteen sets of separate display commands (also known as verb phrases), each with its own sort conditions. In order to display all of the information, a meaningful relationship has to exist among the separate sort condition sets. The following rules apply:

- Up to sixteen display commands and their associated sort conditions can be used. The first display command does not have to have any sort condition. Only the last display command may be a detail command, such as PRINT or LIST; other preceding display commands must be aggregating commands.
- WHERE and IF criteria apply to the records selected for the report as a whole. WHERE and IF criteria are explained in Chapter 5, *Selecting Records for Your Report*.
- When a sort phrase is used with a display command, the display commands following it must use the same sorting condition in the same order. For example:

```
TABLE FILE EMPLOYEE
SUM ED_HRS
SUM CURR_SAL CNT.CURR_SAL BY DEPARTMENT
PRINT LAST_NAME AND FIRST_NAME BY DEPARTMENT
END
```

The first SUM does not have a sort condition. The second SUM has a sort condition: BY DEPARTMENT. Because of this sort condition, the PRINT command must have the BY DEPARTMENT sort condition also.

Example **Using Multiple Display and Sort Fields**

The following request summarizes several levels of detail in the data source:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
SUM CURR_SAL BY DEPARTMENT
SUM CURR_SAL BY DEPARTMENT BY LAST_NAME
END
```

The command SUM CURR_SAL calculates the total amount of current salaries; SUM CURR_SAL BY DEPARTMENT calculates the total amounts of current salaries in each department; SUM CURR_SAL BY DEPARTMENT BY LAST_NAME calculates the total amounts of current salaries for each employee name.

The output is:

NUMBER OF RECORDS IN TABLE= 12 LINES= 12

PAGE 1

CURR_SAL	DEPARTMENT	CURR_SAL	LAST_NAME	CURR_SAL
-----	-----	-----	-----	-----
\$222,284.00	MIS	\$188,882.00	BLACKWOOD	\$21,788.00
			CROSS	\$27,862.00
			GREENSPAN	\$9,888.00
			JONES	\$18,488.00
			MCCOY	\$18,488.00
			SMITH	\$13,288.00
	PRODUCTION	\$114,282.00	BANNING	\$29,788.00
			IRVING	\$26,862.00
			MCKNIGHT	\$16,188.00
			ROMANS	\$21,128.00
			SMITH	\$9,588.00
			STEVENS	\$11,888.00

Improving Efficiency With External Sorts

When a report is generated, by default it is sorted using an internal sorting procedure. This sorting procedure is optimized for reports of up to approximately 5000 records (for example, up to 5000 lines of a tabular report).

You can generate larger reports—exceeding 5000 records—up to 75% faster by using dedicated sorting products, such as SyncSort or DFSORT.

To use an external sort, the EXTSORT SET parameter must be ON (the default).

Note that external sorting is supported with the French, Spanish, German, and Scandinavian National Languages. To specify the National Language Support Environment, use the LANG parameter as described in the *Developing Applications* manual.

Reference

Requirements for External Sorting

You can use the following external sort products with any TABLE, FML, GRAPH, or MATCH request:

- **MVS.** DFSORT and SyncSort.
- **CMS.** DFSORT, SyncSort, and VMSORT.

Under CMS, the sort must be 31-bit addressable.

We recommended that you issue a GLOBAL TXTLIB command to identify the location of the sort software. If the sort program cannot be located, a GLOBAL TXTLIB SORTLIB command is issued. If it still cannot locate the sort software, it will terminate with the following error message:

```
(FOC909) CRITICAL ERROR IN EXTERNAL SORT. RETURN CODE IS:16
```

You will also receive an error message in CMS if your system sort is DFSORT/CMS and your temporary disk is not large enough for your sort requirements. With DFSORT/CMS, the maximum temporary disk that can be obtained for a 3380 device is 16 cylinders. If this is not large enough for your sort requests, you may need to modify the DUFMAC MACRO to allow more space for sort work files. Consult the *IBM DFSORT/CMS* manual for your release of DFSORT.

Your sort product's installation parameters are always respected.

Procedure How a Specific Sort Is Chosen

To determine which sort is used, the following criteria are evaluated, in this sequence:

1. **BINS.** If an entire report can be sorted within the work area (BINS), the external sort is not invoked, even if EXTSORT is set ON.
2. **EXTERNAL.** If BINS is not large enough to sort the entire report and EXTSORT is set ON, approximately the first 5000 records are sorted internally and the remaining records are passed to the external sort utility.

Syntax How to Control External Sorting

You can turn the external sorting feature on and off using the SET EXTSORT command

```
SET EXTSORT = {ON|OFF}
```

where:

ON

Enables the selective use of a dedicated external sorting product to sort reports. This value is the default.

OFF

Uses the internal sorting procedure to sort all reports.

Syntax How to Query the Sort Type

To determine which sort is being used for a given report, issue the following command after the report request:

```
? STAT
```

The command displays the following values for the SORT USED parameter:

FOCUS

The internal sorting procedure was used to sort the entire report.

SQL

You are using a relational data source and the RDBMS sorted the report.

EXTERNAL

An external sorting product sorted the report.

Report optimization determines the scope of the external sort. The internal sorting procedure is used to sort the first group of approximately 5000 records, and the external sort handles the remainder.

To confirm the extent of the external sort, check the ? STAT command's INTERNAL MATRIX CREATED parameter. If the external sort handles the entire report, the parameter displays NO; if the first group of records is sorted internally and the external sort handles the remainder, it displays YES.

NONE

The report did not require sorting.

Aggregation by External Sort

External sorts can be used to perform aggregation with a significant decrease in processing time in comparison to using the internal sort facility. The gains are most notable with relatively simple requests against large data sources.

When aggregation is performed by an external sort, the statistical variables &RECORDS and &LINES are equal because the external sort products do not return a line count for the answer set. This is a behavior change and affects any code that checks the value of &LINES.

Syntax

How to Use Aggregation in Your External Sort

```
SET EXTAGGR = aggropt
```

where:

aggropt

Can be one of the following:

OFF disallows aggregation by an external sort.

NOFLOAT allows aggregation if there are no floating point data fields present.

ON allows aggregation by an external sort. This value is the default.

Reference

Usage Notes for Aggregating With an External Sort

- You must be using SYNCSORT or DFSORT.
- EXTAGGR cannot be set to OFF.
- Your query should be simple (that is, it should be able to take advantage of the TABLEF facility). For related information, see Chapter 15, *Improving Report Processing*.
- The PRINT display command may not be used in the query.
- SET ALL must be equal to OFF.
- Only the following column prefixes are allowed: SUM, AVG, CNT, FST.
- Columns can be calculated values or have a row total.
- CMS DFSORT does not support aggregation of numeric data types. When SET EXTAGGR = NOFLOAT and your query aggregates numeric data, the external sort is not called and aggregation is performed through the internal sorting procedure.

Example **Changing Output by Using External Sorts for Aggregation**

If you use SUM on an alphanumeric field in your report request without using an external sort, the last instance of the sorted fields is displayed in the output. Turning on aggregation in the external sort displays the first record instead.

The following command turns aggregation ON:

```
SET EXTAGGR = ON
TABLE FILE CAR
SUM CAR BY COUNTRY
END
```

The output is:

COUNTRY	CAR
-----	----
ENGLAND	JAGUAR
FRANCE	PEUGEOT
ITALY	ALFA ROMEO
JAPAN	DATSUN
W GERMANY	AUDI

The following command turns aggregation OFF:

```
SET EXTAGGR = OFF
TABLE FILE CAR
SUM CAR BY COUNTRY
END
```

The output is:

COUNTRY	CAR
-----	----
ENGLAND	TRIUMPH
FRANCE	PEUGEOT
ITALY	MASERATI
JAPAN	TOYOTA
W GERMANY	BMW

Note: The SET SUMPREFIX command in conjunction with aggregation using an external sort also affects the order of information displayed in your report. SUMPREFIX is described in *Changing Retrieval Order With Aggregation* on page 4-37.

Changing Retrieval Order With Aggregation

When an external sort product performs aggregation of alphanumeric or smart date formats, the order of the answer set returned differs from the order of the internally sorted answer sets.

External sort products return the first alphanumeric or smart date record that was aggregated. Conversely, internal sorting returns the last record.

The SUMPREFIX command allows users to choose the answer set display order.

Syntax

How to Set Retrieval Order

```
SET SUMPREFIX = {LST|FST}
```

where:

LST

Displays the last value when alphanumeric or smart date data types are aggregated. This value is the default.

FST

Displays the first value when alphanumeric or smart date data types are aggregated.

Using External Sorts to Extract Data

You can use external sorts to create HOLD extract files, producing savings of up to twenty percent in processing time. The gains are most notable with relatively simple requests against large data sources.

Syntax

How to Create HOLD FILES With an External Sort

```
SET EXTHOLD = {OFF|ON}
```

where:

OFF

Disables HOLD files by an external sort.

ON

Enables HOLD files by an external sort. This value is the default.

Reference

Usage Notes for Creating HOLD Files With External Sorts

- The default setting of EXTSORT=ON must be in effect.
- EXTHOLD must be ON.
- The request must contain a BY field.
- The request must contain ON TABLE HOLD or ON TABLE HOLD AS.
- Your query should be simple. AUTOTABLEF analyzes a query and determines whether the combination of display commands and formatting options require the internal matrix. In cases where it's determined that a matrix is not necessary to satisfy the query we avoid the extra internal costs associated with creating the matrix. The internal matrix is stored in a file or data set named FOCSORT. Its default is ON so that performance gains may be realized.
- SET ALL must be OFF.
- There cannot be an IF/WHERE TOTAL or BY TOTAL in the request.
- If a request contains a SUM command, EXTAGGR must be set ON and the only column prefixes allowed are SUM. and FST.
- If a request contains a PRINT command, the column prefixes allowed are MAX., MIN., FST., and LST.

Estimating SORTWORK Sizes for an External Sort

Usually the size of your SORTWORK files is determined when an external sort product is installed. Large queries that use external sorts can rapidly exhaust SORTWORK space, resulting in FOC909 errors. This problem is solved if the sorts include the parameter 'FILSZ=En' when invoked. This parameter enables the sorting algorithms to estimate SORTWORK space requirements for each sort parameter request. ESTRECORDS is used to pass the estimated number of records to be sorted in the request. In order to make an accurate estimate for your ESTRECORDS setting, we suggest that you run the report without an external sort in order to get a record count.

Syntax**How to Provide an Estimate of Input Records for Sorting**

```
ON TABLE SET ESTRECORDS n
```

where:

n

Is the estimated number of records to be sorted.

Note:

- ESTRECORDS can only be set with the ON TABLE SET command within the TABLE, MATCH, or GRAPH request.
- For CMS/SyncSort the 'FILSZ=*En*' parameter is ignored. Therefore, SET ESTRECORDS *n* has no effect.
- If an attempt is made to SET ESTRECORDS from the command line, FOCPARM, or PROFILE FOCEXEC, the following error is generated:

```
SET ESTRECORDS = n
```

```
(FOC36210) THE SPECIFIED PARAMETER CAN ONLY BE SET ON TABLE: ESTRECORDS
```

Displaying External Sort Messages

By default, error messages created by your external sort product are not displayed. However, you may wish to display these messages on your screen for diagnostic purposes. See *How to Display Messages for DFSORT and SyncSort (MVS)* on page 4-39.

Procedure**How to Display Messages for DFSORT and SyncSort (MVS)**

To display DFSORT messages:

1. Create a sequential file with these attributes:

```
DCB=(LRECL=80,RECFM=F,BLKSIZE=80)
```

2. Create the following record beginning in column 2

```
OPTION MSGPRT=ALL,MSGDDN=trace_name
```

where *trace_name* is any user-defined ddname.

3. Allocate the file to ddname DFSPARM.
4. Allocate the ddnames *trace_name* and SORTDIAG to a single file or to SYSOUT. This ensures that all trace and diagnostic messages are written to the same file.

To display SyncSort messages:

1. Create a sequential file with these attributes:

```
DCB=( LRECL=80,RECFM=F,BLKSIZE=80 )
```

2. Create the following record beginning in column 2:

```
MSG=AB,BMSG,MSGDD=trace_name,LIST
```

where *trace_name* is any user-defined ddname.

3. Allocate the file to ddname \$ORTPARM.
4. Allocate the ddname *trace_name* to a file or to SYSOUT.

CHAPTER 5

Selecting Records for Your Report

Topics:

- Choosing a Filtering Method
- Selections Based on Individual Values
- Selection Based on Aggregate Values
- Using Compound Expressions for Record Selection
- Using Operators in Record Selection Tests
- Types of Record Selection Tests
- Selections Based on Group Key Values
- Setting Limits on the Number of Records Read
- Selecting Records Using IF Phrases
- Reading Selection Values From a File
- Assigning Screening Conditions to a File
- VSAM Record Selection Efficiencies

When generating a report and specifying which fields to display, you may not want to display every instance of a field. By including selection criteria, you can display only those field values that meet your needs. In effect, you can select a subset of the data—a subset that you can easily redefine each time you issue the report request.

When developing a report request, you can define criteria that select records based on a variety of factors:

- The values of an individual field. See *Selections Based on Individual Values* on page 5-2.
- The aggregate value of a field (for example, the sum or average of a field's values). See *Selection Based on Aggregate Values* on page 5-10.
- The existence of missing values for a field; whether a field's values fall within a range; whether a field does not contain a certain value. See *Types of Record Selection Tests* on page 5-16.
- The number of records that exist for a field—for example, the first 50 records—rather than on the field's values. See *Setting Limits on the Number of Records Read* on page 5-27.
- For non-FOCUS data sources that have group keys, you can select records based on group key values. See *Selections Based on Group Key Values* on page 5-26.

In addition, you can take advantage of a variety of record selection efficiencies, including assigning filtering criteria to a data source and reading selection values from a file.

Choosing a Filtering Method

There are two phrases for selecting records: WHERE and IF. We recommend that you use WHERE to select records; IF offers a subset of the functionality of WHERE. Everything that you can accomplish with IF, you can also accomplish with WHERE; WHERE can accomplish things that IF cannot.

If you used IF to select records in the past, remember that WHERE and IF are two different phrases, and may require different syntax to achieve the same result.

WHERE syntax is described and illustrated throughout this topic. For details on IF syntax, see *Selecting Records Using IF Phrases* on page 5-28.

Selections Based on Individual Values

The WHERE phrase selects the data source records to be included in a report. The data is evaluated according to the selection criteria before it is retrieved from the data source.

You can use as many WHERE phrases as necessary to define your selection criteria. For an illustration, see *Using Multiple WHERE Phrases* on page 5-4; for additional information, see *Using Compound Expressions for Record Selection* on page 5-12.

Note: Multiple selection tests on fields that reside on separate paths of a multi-path data source are processed as though connected by either AND or OR operators, based on the setting of a parameter called MULTIPATH. For details, see *Controlling Record Selection in Multi-Path Data Sources* on page 5-5.

Syntax

How to Select Records With WHERE

The basic selection syntax using WHERE is

```
WHERE criteria [;]
```

where:

criteria

Are the criteria for selecting records to include in the report. The criteria must be defined in a valid expression that evaluates as true or false (that is, a Boolean expression). Expressions are described in detail in Chapter 8, *Using Expressions*. Operators that can be used in WHERE expressions—such as CONTAINS, IS, and GT—are described in *Operators Supported for WHERE and IF Tests* on page 5-13.

;

An optional semicolon can be used to enhance the readability of the request. It does not affect the report.

Reference

Usage Notes for WHERE Phrases

The WHERE phrase can include:

- Most expressions that would be valid on the right-hand side of a DEFINE expression. However, the logical expression IF ... THEN ... ELSE cannot be used.
- Real fields, temporary fields, and fields in joined files.
- The operators EQ, NE, GE, GT, LT, LE, CONTAINS, OMITS, FROM ... TO, NOT-FROM ... TO, INCLUDES, EXCLUDES, LIKE, and NOT LIKE.
- All arithmetic operators (+, -, *, /, **), as well as functions (MIN, MAX, ABS, and SQRT, for example).
- An alphanumeric expression, which can be a literal, or a function yielding an alphanumeric or numeric result using EDIT or DECODE.
Note that files used with DECODE expressions can contain two columns, one for field values and one for numeric decode values.
- Alphanumeric and date literals enclosed in single quotation marks and date-time literals in the form DT (*date-time literal*).
- All functions.

You can build complex selection criteria by joining simple expressions with AND and OR logical operators and, optionally, adding parentheses to explicitly specify the order of evaluation. This is easier than trying to achieve the same effect with the IF phrase, which may require the use of a separate DEFINE command. For details, see *Using Compound Expressions for Record Selection* on page 5-12.

Reference

Selecting Records for Partitioned FOCUS Data Sources

When you are reporting from a partitioned FOCUS data source, if your selection criteria are based on the same fields used to place data in the partitions, only those partitions with relevant data will be opened for retrieval. For details on partitioned FOCUS data sources, see the *Describing Data* manual.

Example **Using a Simple WHERE Test**

To show only the names and salaries of employees earning more than \$20,000 a year, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME AND CURR_SAL
BY LAST_NAME NOPRINT
WHERE CURR_SAL GT 20000
END
```

In this example, CURR_SAL is a selected field, and CURR_SAL GT 20000 is the selection criterion. Only those records with a current salary greater than \$20,000 are retrieved and displayed; all other records are ignored.

The output is:

PAGE 1

LAST_NAME	FIRST_NAME	CURR_SAL
BANNING	JOHN	\$29,700.00
BLACKWOOD	ROSEMARIE	\$21,780.00
CROSS	BARBARA	\$27,062.00
IRVING	JOAN	\$26,862.00
ROMANS	ANTHONY	\$21,120.00

Example **Using Multiple WHERE Phrases**

You can use as many WHERE phrases as necessary to define your selection criteria. For example, this request uses multiple WHERE phrases.

```
TABLE FILE EMPLOYEE
PRINT EMP_ID LAST_NAME
WHERE SALARY GT 20000
WHERE DEPARTMENT IS 'MIS' OR 'PRODUCTION'
WHERE LAST_NAME IS 'CROSS' OR 'BANNING'
END
```

The output is:

PAGE 1

EMP_ID	LAST_NAME
119329144	BANNING
818692173	CROSS

For related information, see *Using Compound Expressions for Record Selection* on page 5-12.

Controlling Record Selection in Multi-Path Data Sources

When you report from a multi-path data source, a parent segment may have children down some paths but not others. The MULTIPATH parameter allows you to control whether such a parent segment is omitted from the report output.

The MULTIPATH setting also affects the processing of selection tests on independent paths. If MULTIPATH is set to COMPOUND, WHERE or IF tests on separate paths are treated as if they are connected by an AND operator. That is, *all* paths must pass the screening tests in order for the parent to be included in the report output.

With MULTIPATH set to SIMPLE, WHERE or IF tests on separate paths are considered independently, as if an OR operator connected them. Therefore, a parent instance is included in the report if at least one of the paths passes its screening test. A warning message is produced, indicating that if the request contains a test on one path, data is also retrieved from another, independent path. Records on the independent path are retrieved regardless of whether the condition is satisfied on the tested path.

The MULTIPATH settings apply in all types of data sources and in all reporting environments (TABLE, TABLEF, MATCH, GRAPH, requests with multiple display commands). MULTIPATH also works with alternate views, indexed views, filters, DBA, and joined structures.

Syntax

How to Control Record Selection in Multi-path Data Sources

To set MULTIPATH from the command level or in a stored procedure, the syntax is

```
SET MULTIPATH = {SIMPLE|COMPOUND}
```

To set MULTIPATH in a report request, the syntax is

```
ON TABLE SET MULTIPATH {SIMPLE|COMPOUND}
```

where:

SIMPLE

Includes a parent segment in the report output if:

- It has at least one child that passes its screening conditions.
Note: A unique segment is considered a part of its parent segment and, therefore, does not invoke independent path processing.
- It lacks any referenced child on a path, but the child is optional, as described in *Rules for Determining If a Segment Is Required* on page 5-9.

The (FOC144) warning message is generated when a request screens data in a multi-path report:

```
(FOC144) WARNING. TESTING IN INDEPENDENT SETS OF DATA
```

SIMPLE is the default value for FOCUS for S/390.

COMPOUND

Includes a parent in the report output if it has *all* of its required children. WHERE or IF tests on separate paths are treated as if they are connected by an AND operator. That is, *all* paths must pass the screening tests in order for the parent to be included in the report output.

COMPOUND is the default value for iWay and WebFOCUS.

For related information, see *MULTIPATH and SET ALL Combinations* on page 5-8 and *Rules for Determining If a Segment Is Required* on page 5-9.

Reference

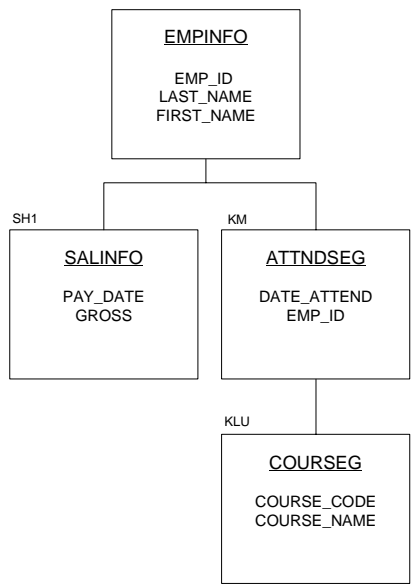
Requirements and Usage Notes for MULTIPATH = COMPOUND

- The minimum memory requirement for the MULTIPATH = COMPOUND setting is 4K per active segment. If there is insufficient memory, the SIMPLE setting is implemented and a message is returned.
There is no limit to the *number* of segment instances (rows); however, no single segment instance can have more than 4K of active fields (referenced fields or fields needed for retrieving referenced fields). If this limit is exceeded, the SIMPLE setting is implemented and a message is returned.
- SET MULTIPATH = COMPOUND creates a pool boundary when reports are pooled.
- WHERE criteria that screen on more than one path with the OR operator are not supported.

Example

Retrieving Data From Multiple Paths

This example uses the following segments from the EMPLOYEE data source:



The request that follows retrieves data from both paths with MULTIPATH = SIMPLE, and displays data if either criterion is met:

```
SET ALL = OFF
SET MULTIPATH = SIMPLE
TABLE FILE EMPLOYEE
PRINT GROSS DATE_ATTEND COURSE_NAME
BY LAST_NAME BY FIRST_NAME
WHERE PAY_DATE EQ 820730
WHERE COURSE_CODE EQ '103'
END
```

The following warning message is generated:

(FOC144) WARNING. TESTING IN INDEPENDENT SETS OF DATA

Although several employees have not taken any courses, they are included in the report output since they have instances on one of the two paths:

LAST_NAME	FIRST_NAME	GROSS	DATE_ATTEND	COURSE_NAME
BLACKWOOD	ROSEMARIE	\$1,815.00	.	.
CROSS	BARBARA	\$2,255.00	.	.
GREENSPAN	MARY	\$750.00	.	.
IRVING	JOAN	\$2,238.50	.	.
JONES	DIANE	\$1,540.00	82/05/26	BASIC REPORT PREP
MCKNIGHT	ROGER	\$1,342.00	.	.
ROMANS	ANTHONY	\$1,760.00	.	.
SMITH	MARY	\$1,100.00	81/11/16	BASIC REPORT PREP
	RICHARD	\$791.67	.	.
STEVENS	ALFRED	\$916.67	.	.

If you run the same request with MULTIPATH = COMPOUND, the employees without instances for COURSE_NAME are omitted from the report output and the warning message is not generated:

LAST_NAME	FIRST_NAME	GROSS	DATE_ATTEND	COURSE_NAME
JONES	DIANE	\$1,540.00	82/05/26	BASIC REPORT PREP FOR PROG
SMITH	MARY	\$1,100.00	81/11/16	BASIC REPORT PREP FOR PROG

Example**MULTIPATH and SET ALL Combinations**

The ALL parameter affects independent path processing. The following table uses examples from the EMPLOYEE data source to explain the interaction of ALL and MULTIPATH:

Request	MULTIPATH	
	SIMPLE	COMPOUND
SET ALL = OFF PRINT EMP_ID PAY_DATE DATE_ATTEND	Shows employees who have <i>either</i> SALINFO data <i>or</i> ATTNDSEG data.	Shows employees who have <i>both</i> SALINFO <i>and</i> ATTNDSEG data.
SET ALL = ON PRINT EMP_ID PAY_DATE DATE_ATTEND	Shows employees who have SALINFO data <i>or</i> ATTNDSEG data <i>or</i> no child data at all.	Same as SIMPLE.
SET ALL = OFF PRINT EMP_ID PAY_DATE DATE_ATTEND WHERE PAY_DATE EQ 980115	Shows employees who have <i>either</i> SALINFO data for 980115 <i>or</i> any ATTNDSEG data. Produces (FOC144) message.	Shows employees who have <i>both</i> SALINFO data for 980115 <i>and</i> ATTNDSEG data.
SET ALL = ON PRINT EMP_ID PAY_DATE DATE_ATTEND WHERE PAY_DATE EQ 980115	Shows employees who have <i>either</i> SALINFO data for 980115 <i>or</i> any ATTNDSEG data. Produces (FOC144) message.	Shows employees who have <i>both</i> SALINFO data for 980115. Any DATE_ATTEND data is also shown.
SET ALL = OFF PRINT ALL.EMP_ID DATE_ATTEND WHERE PAY_DATE EQ 980115	Shows employees who have <i>either</i> SALINFO data for 980115 <i>or</i> any ATTNDSEG data. Produces (FOC144) message.	Shows employees who have SALINFO data for 980115. Any DATE_ATTEND data is also shown.
SET ALL = ON or OFF PRINT EMP_ID PAY_DATE DATE_ATTEND WHERE PAY_DATE EQ 980115 AND COURSE_CODE EQ '103'	Shows employees who have <i>either</i> SALINFO data for 980115 <i>or</i> COURSE 103. Note: SIMPLE treats the AND in the WHERE clause as an OR. Produces (FOC144) message.	Shows employees who have <i>both</i> SALINFO data for 980115 <i>and</i> COURSE 103.

Note: SET ALL = PASS is not supported with MULTIPATH = COMPOUND.

For related information about the ALL parameter, see Chapter 12, *Handling Records With Missing Field Values*.

Reference

Rules for Determining If a Segment Is Required

The segment rule is applied level by level, descending through the data source/view hierarchy. That is, a parent segment's existence depends on the child segment's existence and the child segment depends on the grandchild's existence, and so on for the full data source tree.

The following rules are used to determine if a segment is required or optional:

- When SET ALL is ON or OFF, a segment with WHERE or IF criteria is required for its parent, and all segments up to the root segment are required for their parents.
When SET ALL = PASS, a segment with WHERE or IF criteria is optional.
- IF SET ALL = ON or PASS, all referenced segments with no WHERE or IF criteria are optional for their parents (outer join).
- IF SET ALL = OFF, all referenced segments are required (inner join).
- A referenced segment can become optional if its parent segment uses the ALL. field prefix operator.

Note: ALL = PASS is not supported for all data adapters and, if it is supported, it may behave slightly differently. Check your specific data adapter documentation for detailed information.

For related information about the ALL parameter, see Chapter 12, *Handling Records With Missing Field Values*, and the *Describing Data* manual.

Selection Based on Aggregate Values

You can select records based on the aggregate value of a field—for example, on the sum of a field's values, or on the average of a field's values—by using the **WHERE TOTAL** phrase. **WHERE TOTAL** is very helpful when you employ the aggregate display commands, **SUM** and **COUNT**, and when you use any prefix operator, such as **AVE.** and **PCT.**

In **WHERE** tests, data is evaluated before it is retrieved. In **WHERE TOTAL** tests, however, data is selected after all the data has been retrieved and processed. For an illustration, see *Using WHERE TOTAL for Record Selection* on page 5-11.

Syntax

How to Select Records With WHERE TOTAL

```
WHERE TOTAL criteria[:]
```

where:

criteria

Are the criteria for selecting records to include in the report. The criteria must be defined in a valid expression that evaluates as true or false (that is, a Boolean expression). Expressions are described in detail in Chapter 8, *Using Expressions*. Operators that can be used in **WHERE** expressions—such as **IS** and **GT**—are described in *Operators Supported for WHERE and IF Tests* on page 5-13.

;

An optional semicolon can be used to enhance the readability of the request. It does not affect the report.

Reference

Usage Notes for WHERE TOTAL

- Any reference to a calculated value, or use of a feature that aggregates values, such as **TOT.field**, **AVE.field**, requires the use of **WHERE TOTAL**.
- Fields with prefix operators require the use of **WHERE TOTAL**.
- **WHERE TOTAL** tests are performed at the lowest sort level.
- Alphanumeric and date literals must be enclosed in single quotation marks. Date-time literals must be in the form **DT(date-time literal)**.
- When you use **ACROSS** with **WHERE TOTAL**, data that does not satisfy the selection criteria is represented in the report with the **NODATA** character.
- If you save the output from your report request in a **HOLD** file, the **WHERE TOTAL** test creates a field called **WH\$\$\$T1**, which contains its internal computations. If there is more than one **WHERE TOTAL** test, each **TOTAL** test creates a corresponding **WH\$\$\$T** field and the fields are numbered consecutively.
- The **WHERE TOTAL** test cannot be specified with the **IN (x,y,z)** and **IN FILE** phrases.

Example**Using WHERE TOTAL for Record Selection**

The following example sums current salaries by department.

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
BY DEPARTMENT
END
```

The output is:

PAGE 1

DEPARTMENT	CURR_SAL
-----	-----
MIS	\$108,002.00
PRODUCTION	\$114,282.00

Now, add a WHERE TOTAL phrase to the request in order to generate a report that lists only the departments where the total of the salaries is more than \$110,000.

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
BY DEPARTMENT
WHERE TOTAL CURR_SAL EXCEEDS 110000
END
```

The values for each department are calculated and then each final value is compared to \$110,000. The output is:

PAGE 1

LAST_NAME	CURR_SAL
-----	-----
CROSS	\$27,062.00

Example**Combining WHERE TOTAL and WHERE for Record Selection**

The following request extracts records for the MIS department. Then, CURR_SAL is summed for each employee. If the total salary for an employee is greater than \$20,000, the values of CURR_SAL are processed for the report. In other words, WHERE TOTAL screens data after records are selected.

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
BY LAST_NAME AND BY FIRST_NAME
WHERE TOTAL CURR_SAL EXCEEDS 20000
WHERE DEPARTMENT IS 'MIS'
END
```

The output is:

PAGE 1

LAST_NAME	FIRST_NAME	CURR_SAL
-----	-----	-----
BLACKWOOD	ROSEMARIE	\$21,780.00
CROSS	BARBARA	\$27,062.00

Using Compound Expressions for Record Selection

You can combine two or more simple WHERE expressions, connected by AND and/or OR operators, to create a compound expression.

By default, when multiple WHERE phrases are evaluated, logical ANDs are processed before logical ORs. In compound expressions, you can use parentheses to change the order of evaluation. All AND and OR operators enclosed in parentheses are evaluated first, followed by AND and OR operators outside of parentheses.

This is especially useful when mixing literal OR tests with logical AND and OR tests:

- In a logical AND or OR test, all field names, test relations, and test values are explicitly referenced and connected by the words OR or AND. For example:

```
WHERE (LAST_NAME EQ 'CROSS') OR (LAST_NAME EQ 'JONES')
```

or

```
WHERE (CURR_SAL GT 20000) AND (DEPARTMENT IS 'MIS')  
      AND (CURR_JOBCODE CONTAINS 'A')
```
- In a literal OR test, the word OR is repeated between test values of a field name, but the field name itself and the connecting relational operator are not repeated. For example:

```
WHERE (LAST_NAME EQ 'CROSS' OR 'JONES')
```

Example

Mixing AND and OR Tests

This example illustrates the impact of parentheses on the evaluation of literal ORs and logical ANDs.

In this request, each expression enclosed in parentheses is evaluated first in the order in which it appears. Notice that the first expression contains a literal OR. The result of each expression is then evaluated using the logical AND.

If parentheses had been excluded, the logical AND would have been evaluated before the literal OR.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY LAST_NAME
WHERE (LAST_NAME EQ 'CROSS' OR 'JONES')
AND (CURR_SAL GT 22000)
END
```

The output is:

PAGE	1
LAST_NAME	CURR_SAL
-----	-----
CROSS	\$27,062.00

Using Operators in Record Selection Tests

You can include a variety of operators in your WHERE and IF selection tests. Many of the operators are common for WHERE and IF; however, several are supported only for WHERE tests. For details, see *Operators Supported for WHERE and IF Tests* on page 5-13.

Reference

Operators Supported for WHERE and IF Tests

You can define WHERE and IF selection criteria using the following operators.

WHERE Operator	IF Operator	Meaning
EQ IS	EQ IS	Tests for and selects values equal to the test expression.
NE IS-NOT	NE IS-NOT	Tests for and selects values not equal to the test expression.
GE	GE FROM IS-FROM	Tests for and selects values greater than or equal to the test value (based on the characters 0 to 9 for numeric values, A to Z and a to z for alphanumeric values).
GT EXCEEDS IS-MORE-THAN	GT EXCEEDS IS-MORE-THAN	Tests for and selects values greater than the test value.
LT IS-LESS-THAN	LT IS-LESS-THAN	Tests for and selects values less than the test value.
LE	LE TO	Tests for and selects values less than or equal to the test value.
GE <i>lower</i> AND ... LE <i>upper</i>		Tests for and selects values within a range of values.
LT <i>lower</i> OR ... GT <i>upper</i>		Tests for and selects values outside of a range of values.
FROM <i>lower</i> TO <i>upper</i>		Tests for and selects values within a range of values.
IS-FROM <i>lower</i> TO <i>upper</i>	IS-FROM <i>lower</i> TO <i>upper</i>	Tests for and selects values within a range of values. For WHERE, this is alternate syntax for FROM lower to UPPER; both operators produce identical results.

WHERE Operator	IF Operator	Meaning
NOT-FROM <i>lower</i> TO <i>upper</i>	NOT-FROM <i>lower</i> TO <i>upper</i>	Tests for and selects values that are outside a range of values.
IS MISSING IS-NOT MISSING NE MISSING	IS MISSING IS-NOT MISSING NE MISSING	Tests whether a field contains missing values—that is, if some instances of the field contain no data (have missing data). For information on missing data, see Chapter 12, <i>Handling Records With Missing Field Values</i> .
CONTAINS LIKE	CONTAINS	Tests for and selects values that include a character string matching test value. The string can occur in any position in the value being tested. When used with WHERE, CONTAINS can test alphanumeric fields; when used with IF, it can test both alphanumeric and text fields.
OMITS NOT LIKE	OMITS	Tests for and selects values that do not include a character string matching test value. The string cannot occur in any position in the value being tested. When used with WHERE, OMITS can test alphanumeric fields; when used with IF, it can test both alphanumeric and text fields.
INCLUDES	INCLUDES	Tests whether a chain of values of a given field in a child segment includes all of a list of literals.
EXCLUDES	EXCLUDES	Tests whether a chain of values of a given field in a child segment excludes all of a list of literals.
IN (<i>z,x,y</i>)		Selects records based on values found in an unordered list.
NOT ... IN (<i>z,x,y</i>)		Selects records based on values not found in an unordered list.
IN FILE		Selects records based on values stored in a sequential file.
NOT ... IN FILE		Selects records with field values not found in a sequential file.

Example**Using Operators to Compare a Field to One or More Values**

The following examples illustrate field selection criteria that use one or more values. You may use the operators: EQ, IS, IS-NOT, EXCEEDS, IS-LESS-THAN, and IN.

Example 1: The field LAST_NAME must equal the value JONES:

```
WHERE LAST_NAME EQ 'JONES'
```

Example 2: The field LAST_NAME begins with 'CR' or 'MC':

```
WHERE EDIT (LAST_NAME, '99') EQ 'CR' OR 'MC'
```

Example 3: The field AREA must not equal the value EAST or WEST:

```
WHERE AREA IS-NOT 'EAST' OR 'WEST'
```

Example 4: The value of the field AREA must equal the value of the field REGION:

```
WHERE AREA EQ REGION
```

Note that you cannot compare one field to another in an IF test.

Example 5: The ratio between retail cost and dealer cost must be greater than 1.25:

```
WHERE RETAIL_COST/DEALER_COST GT 1.25
```

Example 6: The field UNITS must be equal to or less than the value 50, and AREA must not be equal to either NORTH EAST or WEST. Note the use of single quotation marks around NORTH EAST. All alphanumeric strings must be enclosed within single quotation marks.

```
WHERE UNITS LE 50 WHERE AREA IS-NOT 'NORTH EAST' OR 'WEST'
```

Example 7: The value of AMOUNT must be greater than 40:

```
WHERE AMOUNT EXCEEDS 40
```

Example 8: The value of AMOUNT must be less than 50:

```
WHERE AMOUNT IS-LESS-THAN 50
```

Example 9: The value of SALES must be equal to one of the numeric values in the unordered list. Use commas or blanks to separate the list values.

```
WHERE SALES IN (43000,12000,13000)
```

Example 10: The value of CAR must be equal to one of the alphanumeric values in the unordered list. Single quotation marks must enclose alphanumeric list values.

```
WHERE CAR IN ('JENSEN', 'JAGUAR')
```

Types of Record Selection Tests

You can select records for your reports using a variety of tests that are implemented using the operators described in *Operators Supported for WHERE and IF Tests* on page 5-13.

You can test for:

- Values that lie within or outside a range. See *Range Tests With FROM and TO* on page 5-16 and *Range Tests With GE and LE or GT and LT* on page 5-17.
- Missing or existing data. See *Missing Data Tests* on page 5-19.
- The existence or absence of a character string. See *Character String Screening With CONTAINS and OMITS* on page 5-19.
- Partially defined character strings in a data field. See *Screening on Masked Fields With LIKE and IS* on page 5-20.
- Literals in a parent segment. See *Qualifying Parent Segments Using INCLUDES and EXCLUDES* on page 5-25.

Range Tests With FROM and TO

Use the operators FROM ... TO and NOT-FROM ... TO in order to determine whether field values fall within or outside a given range. You can use either values or expressions to specify the lower and upper boundaries. Range tests can also be applied on the sort control fields. The range test is specified immediately after the sort phrase.

Syntax

How to Specify a Range Test (FROM and TO)

The syntax for the range test is

```
WHERE [TOTAL] fieldname {FROM|IS-FROM} lower TO upper  
WHERE [TOTAL] fieldname NOT-FROM      lower TO upper
```

where:

fieldname

Is any valid field name or alias.

lower

Are numeric or alphanumeric values or expressions that indicate lower boundaries.
You may add parentheses around expressions for readability.

upper

Are numeric or alphanumeric values or expressions that indicate upper boundaries.
You may add parentheses around expressions for readability.

Example Range Test With FROM ... TO

An example of a range test using expressions as boundaries follows:

```
WHERE SALES FROM (DEALER_COST * 1.4) TO (DEALER_COST * 2.0)
```

Example Range Test With NOT-FROM ... TO

This request displays only employees whose salaries do not fall between \$12,000 and \$22,000:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY LAST_NAME
WHERE CURR_SAL NOT-FROM 12000 TO 22000
END
```

The output is:

PAGE 1

LAST_NAME	CURR_SAL
-----	-----
BANNING	\$29,700.00
CROSS	\$27,062.00
GREENSPAN	\$9,000.00
IRVING	\$26,862.00
SMITH	\$9,500.00
STEVENS	\$11,000.00

Example Range Tests on Sort Fields With FROM ... TO

The following examples demonstrate how to perform range tests when sorting a field using the BY or ACROSS sort phrases:

```
BY MONTH FROM 4 TO 8
```

or

```
ACROSS MONTH FROM 6 TO 10
```

Range Tests With GE and LE or GT and LT

The operators GE (Greater Than or Equal to) and LE (Less Than or Equal to) can be used to specify a range:

- GE ... LE enable you to specify values within the range test boundaries.
- LT ...GT enable you to specify values outside the range test boundaries.

Syntax

How to Specify Range Tests (GE and LE)

To select values that fall within a range, use AND

```
WHERE fieldname GE lower AND fieldname LE upper
```

To find records whose values do not fall in a specified range, use OR

```
WHERE fieldname LT lower OR fieldname GT upper
```

where:

```
fieldname
```

Is any valid field name or alias.

```
lower
```

Are numeric or alphanumeric values or expressions that indicate lower boundaries.
You may add parentheses around expressions for readability.

```
upper
```

Are numeric or alphanumeric values or expressions that indicate upper boundaries.
You may add parentheses around expressions for readability.

Example

Selecting Values Inside a Range

This WHERE phrase selects records in which the UNIT value is between 10,000 and 14,000.

```
WHERE UNITS GE 10000 AND UNITS LE 14000
```

This example is equivalent to:

```
WHERE UNITS GE 10000  
WHERE UNITS LE 14000
```

Example

Selecting Values Outside a Range

This request lists employees whose salaries are either less than \$12,000 or greater than \$22,000.

```
TABLE FILE EMPLOYEE  
PRINT CURR_SAL  
BY LAST_NAME  
WHERE CURR_SAL LT 12000 OR CURR_SAL GT 22000  
END
```

The output is:

PAGE 1

LAST_NAME	CURR_SAL
-----	-----
BANNING	\$29,700.00
CROSS	\$27,062.00
GREENSPAN	\$9,000.00
IRVING	\$26,862.00
SMITH	\$9,500.00
STEVENS	\$11,000.00

Missing Data Tests

When creating report requests, you may want to test for missing data. For this test to be effective, fields that have missing data must have the MISSING attribute set to ON in the Master File. For information on missing data, see Chapter 12, *Handling Records With Missing Field Values* and the *Describing Data* manual.

Syntax

How to Test for Missing Data

The syntax is

```
{WHERE|IF} fieldname {EQ|IS} MISSING
```

where:

fieldname

Is any valid field name or alias.

EQ|IS

Are record selection operators. EQ and IS are synonyms.

Syntax

How to Test for Existing Data

The syntax is

```
{WHERE|IF} fieldname {NE|IS-NOT} MISSING
```

where:

fieldname

Is any valid field name or alias.

NE|IS-NOT

Are record selection operators. NE and IS-NOT are synonyms.

Character String Screening With CONTAINS and OMITS

The CONTAINS and OMITS operators test alphanumeric fields when used with WHERE, and both alphanumeric and text fields when used with IF:

- With CONTAINS, if the characters in the given literal or literals appear anywhere within the characters of the field value, the test is passed.
- OMITS is the opposite of CONTAINS; if the characters of the given literal or literals appear anywhere within the characters of the field's value, the test fails.

CONTAINS and OMITS tests are useful when the exact spelling of a value is not known. As long as you know that a specific string appears within the value, you can retrieve the desired data.

Example

Selecting Records With CONTAINS and OMITS

In the first example, the characters JOHN are contained in JOHNSON and would be selected by the following phrase:

```
WHERE LAST_NAME CONTAINS 'JOHN'
```

The field LAST_NAME may contain the characters JOHN anywhere in the field.

Note that the field name being tested must appear on the left side of the CONTAINS operator.

In the second example, any last name without the string JOHN will be selected:

```
WHERE LAST_NAME OMITS 'JOHN'
```

In the third example, all names that contain the letters ING are displayed.

```
TABLE FILE EMPLOYEE
LIST LAST_NAME AND FIRST_NAME
WHERE LAST_NAME CONTAINS 'ING'
END
```

The output is:

PAGE 1

LIST	LAST_NAME	FIRST_NAME
1	BANNING	JOHN
2	IRVING	JOAN

Screening on Masked Fields With LIKE and IS

A mask is an alphanumeric pattern that you supply for comparison to characters in a data field. The data field must have an alphanumeric format (A).

You can use the LIKE and NOT LIKE or the IS and IS-NOT operators to perform screening on masked fields:

- The LIKE and NOT LIKE operators use the wildcard characters % and _. The percent allows any following sequence of zero or more characters. The underscore indicates that any character in that position is acceptable.
- The IS (or EQ) and IS-NOT (or NE) operators use the wildcard characters \$ and \$*. The dollar sign indicates that any character in that position is acceptable. The \$* combination allows any sequence of zero or more characters. This combination can only be used at the end of the mask.

Note: The LIKE operator is supported in expressions that are used to derive temporary fields with either the DEFINE or the COMPUTE command.

Syntax

How to Screen on Masked Fields (LIKE and NOT LIKE)

To search for records with the LIKE operator, use the syntax

```
WHERE field LIKE 'mask'
```

To reject records based on the mask value, use either

```
WHERE field NOT LIKE 'mask'
```

or

```
WHERE NOT field LIKE 'mask'
```

where:

field

Is any valid field name or alias.

mask

An alphanumeric or text character string you supply. There are two wildcard characters that you can use in the mask: the underscore (_) indicates that any character in that position is acceptable; the percent sign (%) allows any following sequence of zero or more characters.

For related information, see *Restrictions on Masking Characters* on page 5-22.

Syntax

How to Screen on Masked Fields (IS and IS-NOT)

To search for records with the IS operator, use the syntax

```
{WHERE|IF} field {IS|EQ} 'mask'
```

To reject records based on the mask value, use the syntax

```
{WHERE|IF} field {IS-NOT|NE} 'mask'
```

where:

field

Is any valid field name or alias.

IS | IS-NOT

Are record selection operators. EQ is a synonym for IS. NE is a synonym for IS-NOT.

mask

An alphanumeric or text character string you supply. The wildcard characters that you can use in the mask are the dollar sign (\$) and the combination \$*. The dollar sign indicates that any character in that position is acceptable. The \$* combination allows any sequence of zero or more characters.

For related information, see *Restrictions on Masking Characters* on page 5-22.

Reference **Restrictions on Masking Characters**

- The wildcard characters dollar sign (\$) and dollar sign with an asterisk (\$*), which are used with IS operators, are treated as literals with LIKE operators.
- Masking with the characters \$ and \$* is not supported for compound WHERE phrases that use the AND or OR logical operators.

Example **Screening on Initial Characters**

To list all employees who have taken basic-level courses, where every basic course begins with the word BASIC, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT  COURSE_NAME COURSE_CODE
BY LAST_NAME BY FIRST_NAME
WHERE COURSE_NAME LIKE 'BASIC%'
END
```

The output is:

PAGE 1

LAST_NAME	FIRST_NAME	COURSE_NAME	COURSE_CODE
BLACKWOOD	ROSEMARIE	BASIC REPORT PREP NON-PROG	102
CROSS	BARBARA	BASIC REPORT PREP DP MGRS	107
JONES	DIANE	BASIC REPORT PREP FOR PROG	103
SMITH	MARY	BASIC REPORT PREP FOR PROG	103
	RICHARD	BASIC RPT NON-DP MGRS	108

Example **Screening on Characters Anywhere in a Field**

If you want to see which employees have taken a FOCUS course, but you do not know where the word FOCUS appears in the title, bracket the word FOCUS with wildcards (which is equivalent to using the CONTAINS operator):

```
TABLE FILE EMPLOYEE
PRINT  COURSE_NAME COURSE_CODE
BY LAST_NAME BY FIRST_NAME
WHERE COURSE_NAME LIKE '%FOCUS%'
END
```

The output is:

PAGE 1

LAST_NAME	FIRST_NAME	COURSE_NAME	COURSE_CODE
BLACKWOOD	ROSEMARIE	WHAT'S NEW IN FOCUS	202
JONES	DIANE	FOCUS INTERNALS	203

If you want to list all employees who have taken a 20x-series course, and you know that all of these courses have the same code except for the final character, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT  COURSE_NAME COURSE_CODE
BY LAST_NAME BY FIRST_NAME
WHERE COURSE_CODE LIKE '20_'
END
```

The output is:

PAGE 1

LAST_NAME	FIRST_NAME	COURSE_NAME	COURSE_CODE
BLACKWOOD	ROSEMARIE	WHAT'S NEW IN FOCUS	202
JONES	DIANE	FOCUS INTERNALS	203
		ADVANCED TECHNIQUES	201

Example

Screening on Initial Characters and Specific Length

The dollar sign acts as a placeholder in the following example:

```
TABLE FILE EMPLOYEE
PRINT  LAST_NAME
WHERE LAST_NAME IS 'BAN$$$$'
END
```

The output is:

PAGE 1

LAST_NAME

BANNING

The selection criterion in the preceding example retrieves only last names that are seven characters long and whose first three characters are BAN. Characters four through seven can be anything, but the remaining characters (8 through 15) must be blank.

Example

Screening on Records of Unspecified Length

To retrieve records with unspecified lengths, use the dollar sign followed by an asterisk (\$*):

```
WHERE LAST_NAME IS 'BAN$*'
```

This phrase searches for last names that start with the letters BAN, regardless of the name's length. The characters \$* reduce typing, and enable you to define a screen mask without knowing the exact length of the field you wish to retrieve.

Using an Escape Character for LIKE

You can use an escape character in the LIKE syntax to treat the masking characters (%) and _) as literals within the search pattern, rather than as wildcards. This technique enables you to search for these characters in the data. For related information, see *Screening on Masked Fields With LIKE and IS* on page 5-20.

- The escape character itself can be escaped, thus becoming a normal character in a string (for example, 'abc\%\').
- The escape character is only in effect when the ESCAPE syntax is included in the LIKE phrase.
- Every LIKE phrase can provide its own escape character.

Syntax

How to Use the Escape Character

Any single character can be used as an escape character if prefaced with the word ESCAPE.

The syntax is

```
WHERE fieldname LIKE 'mask' ESCAPE 'c'
```

where:

fieldname

Is any valid field name or alias to be evaluated in the selection test.

mask

Is the search pattern that you supply. The single quotation marks are required.

c

Is any single character that you identify as the escape character. If you embed the escape character in the mask, before a % or _, the % or _ character is treated as a literal, rather than as a wildcard. The single quotation marks are required.

Reference

Usage Notes for Escape Characters

- The use of an escape character in front of any character other than %, _, and itself will be ignored.
- Only one escape character can be used per LIKE phrase.
- If a WHERE criterion is used with literal OR phrases, the ESCAPE must be on the first OR phrase and will apply to all subsequent phrases in that WHERE expression. For example:

```
WHERE field LIKE 'ABCg_' ESCAPE 'g' OR 'ABCg%' OR 'g%ABC'
```

Example**Using the Escape Character**

The VIDEOTR2 data source contains an e-mail address field. To search for the e-mail address with the characters 'handy_' you can issue the following request:

```
TABLE FILE VIDEOTR2
PRINT CUSTID LASTNAME FIRSTNAME EMAIL
WHERE EMAIL LIKE 'handy_%'
END
```

Because the underscore character functions as a wildcard character, this request returns two instances, only one of which contains the underscore character.

The output is:

CUSTID	LASTNAME	FIRSTNAME	EMAIL
0944	HANDLER	EVAN	handy_man@usa.com
0944	HANDLER	EVAN	handyman@usa.com

To retrieve only the instance that contains the underscore character you must indicate that the underscore should be treated as a normal character, not a wildcard. The following request retrieves only the instance with the underscore character in the e-mail field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID LASTNAME FIRSTNAME EMAIL
WHERE EMAIL LIKE 'handy\_%' ESCAPE '\'
END
```

The output is:

CUSTID	LASTNAME	FIRSTNAME	EMAIL
0944	HANDLER	EVAN	handy_man@usa.com

Qualifying Parent Segments Using INCLUDES and EXCLUDES

INCLUDES and EXCLUDES work only with multi-segment FOCUS data sources. They test whether instances of a given field in a child segment include or exclude all literals in a list. INCLUDES and EXCLUDES retrieve only parent records. You cannot print or list any field in the same segment as the field specified for the INCLUDES or EXCLUDES test.

Reference**Usage Notes for INCLUDE and EXCLUDES**

- Literals containing embedded blanks must be enclosed in single quotation marks.
- To use more than one INCLUDES or EXCLUDES phrase in a request, begin each phrase on a separate line.
- You can connect the literals you are testing for with ANDs and ORs, however, you cannot create compound INCLUDES and EXCLUDES tests with logical AND and OR operators.

Example

Selecting Records With INCLUDES and EXCLUDES

A request that contains the phrase

```
WHERE JOBCODE INCLUDES A01 OR B01
```

will return employee records with JOBCODE instances for both A01 and B01, as if you had used AND.

In the following example, for a record to be selected its JOBCODE field must have values of both A01 and B01:

```
WHERE JOBCODE INCLUDES A01 AND B01
```

If either one is missing, the record will not be selected for the report.

If the selection criterion is

```
WHERE JOBCODE EXCLUDES A01 AND B01
```

every record that does not have both values is selected for the report.

In the CAR data source, only England produces Jaguars and Jensens, and so the request

```
TABLE FILE CAR
PRINT COUNTRY
WHERE CAR INCLUDES JAGUAR AND JENSEN
END
```

generates this output:

```
PAGE      1
```

```
COUNTRY
-----
ENGLAND
```

Selections Based on Group Key Values

Certain types of non-FOCUS data sources use group keys. A group key is a single key composed of several fields. You can use a group name to refer to a group key's fields.

To select records based on a group key value, you need to supply the value of each field. The values must be separated by the slash character (/).

Note that a WHERE phrase that refers to a group field cannot be used in conjunction with AND or OR. For related information see *Using Compound Expressions for Record Selection* on page 5-12.

Example

Selecting Records Using Group Keys

Suppose that a data source has a group key named PRODNO, which contains three separate fields. The first is stored in alphanumeric format, the second as a packed decimal, the third as an integer. A screening phrase on this group might be:

```
WHERE PRODNO EQ 'RS/62/83'
```

For details on working with non-FOCUS data sources, see the *Describing Data* manual.

Setting Limits on the Number of Records Read

For some reports, a limited number of records is satisfactory. When the specified number of records is retrieved, record retrieval can stop. This is useful when:

- You are designing a new report, and you need only a few records from the actual data source to test your design.
- The database administrator needs to limit the size of reports by placing an upper limit on retrieval from very large data sources. This limit is attached to the user's password.
- You know the number of records that meet the test criteria. You can specify that number so that the search does not continue beyond the last record that meets the criteria. For example, suppose only ten employees use electronic transfer of funds and you want to retrieve only those records. The record limit would be ten, and retrieval would stop when the tenth record is retrieved. The data source will not be searched any further.

Syntax

How to Limit the Number of Records Read

There are two ways to limit the number of records retrieved. You can use the syntax

`WHERE RECORDLIMIT EQ n`

where:

n

Is a number greater than 0, and indicates the number of records to be retrieved. This syntax can be used with FOCUS and non-FOCUS data sources.

For all non-FOCUS data sources, you can also use the syntax

`WHERE READLIMIT EQ n`

where:

n

Is a number greater than 0, and indicates the number of read operations (not records) to be performed. For details, see the appropriate data adapter manual.

Tip:

If an attempt is made to apply the READLIMIT test to a FOCUS data source, the request is processed correctly, but the READLIMIT phrase is ignored.

Example

Limiting the Number of Records Read

The following request retrieves four records, generating a four-line report:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME AND EMP_ID
WHERE RECORDLIMIT EQ 4
END
```

The output is:

PAGE 1

LAST_NAME	FIRST_NAME	EMP_ID
STEVENS	ALFRED	071382660
SMITH	MARY	112847612
JONES	DIANE	117593129
SMITH	RICHARD	119265415

Selecting Records Using IF Phrases

The IF phrase selects records to be included in a report, and offers a subset of the functionality of WHERE. For a list of supported IF operators, see *Using Operators in Record Selection Tests* on page 5-13.

Tip:

Unless you specifically require IF syntax (for example, to support legacy applications), we recommend using WHERE.

Syntax

How to Select Records Using the IF Phrase

The syntax for record selection using the IF phrase is

```
IF fieldname operator literal [OR literal]
```

where:

fieldname

Is the field you want to test (the test value).

operator

Is the type of selection operator you want. Valid operators are described in *Operators Supported for WHERE and IF Tests* on page 5-13.

literal

Can be the MISSING keyword (as described in *Missing Data Tests* on page 5-19) or alphanumeric or numeric values that are in your data source, with the word OR between values.

Note that all literals that contain blanks (for example, New York City) and all date and date-time literals must be enclosed within single quotation marks.

Note: The IF phrase alone cannot be used to create compound expressions by connecting simple expressions with AND and OR logical operators. Compound logic requires that the IF phrase be used with the DEFINE command, as described in Chapter 8, *Using Expressions*. You can accomplish this more easily with WHERE. See *Using Compound Expressions for Record Selection* on page 5-12.

Example**Using Multiple IF Phrases**

You can use as many IF phrases as necessary to define all your selection criteria, as illustrated in the following example:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID LAST_NAME
IF SALARY GT 20000
IF DEPARTMENT IS MIS
IF LAST_NAME IS CROSS OR BANNING
END
```

All of these criteria must be satisfied in order for a record to be included in a report. The output is:

PAGE 1

EMP_ID	LAST_NAME
818692173	CROSS

Reading Selection Values From a File

Instead of typing literal test values in a WHERE or IF phrase, you can store them in a file and refer to the file in the report request. You can then select records based on equality—or inequality—tests on values stored in the file.

This method has the following advantages:

- You can save time by coding a large set of selection values once, then using these values as a set in as many report requests as you wish. You also ensure consistency by maintaining the criteria in just one location.
- If the selection values already exist in a data source, you can quickly create a file of selection values by generating a report and saving the output in a HOLD or SAVE file. You can then read selection values from that file.

If you use a HOLD file, it must either be in BINARY format (the default) or in ALPHA (simple character) format; if you use a SAVE file, it must be in ALPHA format (the default). You can also use a SAVB file if the selection values are alphanumeric. See Chapter 11, *Saving and Reusing Report Output*, for information on HOLD and SAVE files.

Note that in MVS, a HOLD file in BINARY format that is used for selection values must be allocated to ddname HOLD (the default); the other extract files used for this purpose can be allocated to any ddname.

- You can include entries with mixed-case and special characters.

Syntax

How to Read Selection Values From a File (WHERE)

The syntax for using WHERE to select records based on values in a file is

`WHERE [NOT] fieldname IN FILE file`

where:

fieldname

Is the name of the selection field. It can be any real or temporary field in the data source being reported on.

file

Is the name of the file.

For MVS, this is the ddname assigned by a DYNAM or TSO ALLOCATE command. On CMS, the ddname is assigned by a FILEDEF command.

For related information, see *Usage Notes for Reading Values From a File* on page 5-31.

Syntax

How to Read Selection Values From a File (IF)

The syntax for using IF to select records based on values in a file is

`IF fieldname operator (file) [OR (file) ...]`

where:

fieldname

Is any valid field name or alias.

operator

Is the EQ, IS, NE, or IS-NOT operator (see *Operators Supported for WHERE and IF Tests* on page 5-13).

file

Is the name of the file.

For MVS, this is the ddname assigned by a DYNAM or TSO ALLOCATE command.

For CMS, this is the ddname assigned by a FILEDEF command.

For related information, see *Usage Notes for Reading Values From a File* on page 5-31.

Reference

Usage Notes for Reading Values From a File

In order to read selection criteria from a file, the file must comply with the following rules:

- Each value in the file must be on a separate line.
For IF, more information can appear on a line, but only the first data value encountered on the line is used.
- The selection value must start in column one.
- The values are assumed to be in character format, unless the file name is HOLD, and numeric digits are converted to internal computational numbers where needed (for example, binary integer).
- For IF, the total of all files can be up to 32,767 literals, including new line and other formatting characters. Lower limits apply to fixed sequential and other non-relational data sources.
- For WHERE, the file can be approximately 16,000 bytes. If the file is too large, an error message displays.
- For WHERE, alphanumeric values with embedded blanks or any mathematical operator (-, +, *, /) must be enclosed in single quotation marks.
- For WHERE, when a compound WHERE phrase uses IN FILE more than once, the specified files must have the same record formats.
If your list of literals is too large, an error is displayed.
- For IF, sets of file names may be used, separated by the word OR. Also, actual literals may be mixed with the file names. For example:

```
IF fieldname operator (filename) OR literal...etc...
```

Example

Reading Selection Values From a File (WHERE)

Create a file named EXPER, which contains the values B141 and B142.

This request uses selection criteria from the file EXPER. All records for which PRODUCT_ID has a value of B141 or B142 will be selected:

```
TABLE FILE GGPRODS
SUM UNIT_PRICE
BY PRODUCT_DESCRIPTION
WHERE PRODUCT_ID IN FILE EXPER
END
```

If you include the selection criteria directly in the request, the WHERE phrase would specify the values explicitly:

```
WHERE PRODUCT_DESCRIPTION EQ 'B141' or 'B142'
```

The output is:

Product	Unit Price
French Roast	81.00
Hazelnut	58.00

Example **Reading Selection Values From a File (IF)**

The following request

```
TABLE FILE GGPRODS
SUM UNIT_PRICE
BY PRODUCT_DESCRIPTION
IF PRODUCT_ID IS (EXPER)
END
```

together with the EXPER file containing the following records

```
B141
B142
```

generates this output:

Product	Unit Price
-----	-----
French Roast	81.00
Hazelnut	58.00

The value of PRODUCT_ID is compared to the values in the EXPER file for the selection criterion.

Assigning Screening Conditions to a File

You can assign screening conditions to a data source, independent of a request, and activate these screening conditions for use in report requests against the data source.

A filter is a packet of definitions that resides at the file level, containing WHERE and/or IF criteria. Whenever a report request is issued against a data source, all filters that have been activated for that data source are in effect. WHERE or IF syntax that is valid in a report request is also valid in a filter.

A filter can be declared at any time before the report request is run. The filters are available to subsequent requests during the session in which the filters have been run. For details, see *How to Declare a Filter* on page 5-33.

Filters allow you to:

- Declare a common set of screening conditions that apply each time you retrieve data from a data source. You can declare one or more filters for a data source.
- Declare a set of screening conditions and dynamically turn them on and off.
- Restrict access to data without specifying rules in the Master File.

In an interactive environment, filters also reduce repetitive ad hoc typing.

Note: Simply declaring a filter for a data source does *not* make it active. A filter must be activated with a SET command. For details, see *How to Activate or Deactivate Filters* on page 5-35.

Syntax

How to Declare a Filter

A filter can be described by the following declaration

```

FILTER FILE filename [CLEAR|ADD]
  [filter-defines;]
  NAME=filtername1 [,DESC=text]
  where-if phrases
  .
  .
  .
  NAME=filternamen [,DESC=text]
  where-if phrases
END

```

where:

filename

Is the name of the Master File to which the filters apply.

ADD/CLEAR

ADD enables you to add new filter phrases to an existing filter declaration, without clearing previously defined filters.

CLEAR deletes any existing filter phrases, including any previously defined virtual fields.

filter-defines

Are virtual fields declared for use in filters. For more information, see *Usage Notes for Virtual Fields Used in Filters* on page 5-34.

filtername1...filternamen

Is the name by which the filter is referenced in subsequent SET FILTER commands. This name may be up to eight characters and must be unique for a particular file name.

text

Describes the filter for documentation purposes. Text must fit on one line.

where-if phrases

Are screening conditions that can include all valid syntax. They may refer to data source fields and virtual fields in the Master File; they may not refer to virtual fields declared using a DEFINE command. They may not refer to other filter names.

Reference

Usage Notes for Virtual Fields Used in Filters

Virtual fields used in filters:

- Are exclusively local to (or usable by) filters in a specific filter declaration.
- Cannot be referenced in a DEFINE or TABLE command.
- Support any syntax valid for virtual fields in a DEFINE command.
- Cannot reference virtual fields in a DEFINE command, but can reference virtual fields in the Master File.
- Do not count toward the display field limit, unlike virtual fields in DEFINE commands.
- Must all be declared before the first named filter.
- Must each end with a semi-colon.
- Cannot be enclosed between the DEFINE FILE and END commands.

Example

Declaring Filters

The first example creates the filter named UK, which consists of one WHERE condition. It also adds a definition for the virtual field MARK_UP to the set of virtual fields already being used in filters for the CAR data source.

When a report request is issued for CAR, with UK activated, the condition WHERE MARK_UP is greater than 1000 is automatically added to the request.

Note: The virtual field MARK_UP cannot be explicitly displayed or referenced in the TABLE request.

```
FILTER FILE CAR ADD
MARK_UP/D7=RCOST-DCOST;
NAME=UK
WHERE MARK_UP GT 1000
END
```

The second example declares three named filters for the CAR data source: ASIA, UK, and LUXURY. The filter ASIA contains a textual description, for documentation purposes only. CLEAR, on the first line, erases any previously existing filters for CAR, as well any previously defined virtual fields used in filters for CAR, before it processes the new definitions.

```
FILTER FILE CAR CLEAR
NAME=ASIA,DESC=Asian cars only
IF COUNTRY EQ JAPAN
NAME=UK
IF COUNTRY EQ ENGLAND
NAME=LUXURY
IF RETAIL_COST GT 50000
END
```


Syntax

How to Activate or Deactivate Filters

Filters can be activated and deactivated with the command

```
SET FILTER= {z|xx[ yy zz]} IN file {ON|OFF}
```

where:

Denotes all declared filters (default).

xx, yy, zz

Are the names of filters as declared in the NAME = syntax of the FILTER FILE command.

file

Is the name of the data source you are assigning screening conditions to.

ON|OFF

ON activates all (*) or specifically named filters for the data source. The maximum number of filters you can activate for a data source is limited by the number of WHERE/IF phrases the filters contain, not to exceed the limit of WHERE/IF criteria in any single report request.

OFF deactivates (*) or specifically named filters for the data source. This value is the default.

Note: The SET FILTER command is limited to one line. To activate more filters than fit on one line, issue additional SET FILTER commands. As long as you specify ON, the effect is cumulative.

Example

Activating and Deactivating Filters

The following commands activate A, B, C, D, E, F and deactivate G (assuming that it was set ON previously):

```
SET FILTER = A B C IN CAR ON
SET FILTER = D E F IN CAR ON
SET FILTER = G IN CAR OFF
```

The following commands activate some filters and deactivate others:

```
SET FILTER = UK LUXURY IN CAR ON
...
TABLE FILE CAR
PRINT COUNTRY MODEL RETAIL_COST
END
...
SET FILTER = LUXURY IN CAR OFF
TABLE FILE CAR
PRINT COUNTRY MODEL RETAIL_COST
END
```

The first SET FILTER command activates the filters UK and LUXURY, assigned to the CAR data source, and applies their screening conditions to any subsequent report request against the CAR data source.

The second SET FILTER command deactivates the filter LUXURY for the CAR data source. Unless LUXURY is reactivated, any subsequent report request against CAR will not apply the conditions in LUXURY, but will continue to apply UK.

Syntax

How to Query the Status of Filters

You can determine the status of existing filters using the syntax

```
? FILTER [{file|*}] [SET] [ALL]]
```

where:

file

Is the name of a Master File.

*

Displays filters for all Master Files for which filters have been declared.

SET

Displays only active filters.

ALL

Displays all information about the filter, including its description and the exact WHERE/IF definition.

Example

Querying Filters

To query filters, issue the following command:

```
FILTER FILE CAR CLEAR
NAME=BOTH, DESC=Asian and British cars only
IF COUNTRY EQ JAPAN AND ENGLAND
END
SET FILTER =BOTH IN CAR ON
TABLE FILE CAR
PRINT CAR RETAIL_COST
BY COUNTRY
END
```

The output is:

COUNTRY	CAR	RETAIL_COST
ENGLAND	JAGUAR	8,878
	JAGUAR	13,491
	JENSEN	17,850
	TRIUMPH	5,100
JAPAN	DATSUN	3,139
	TOYOTA	3,339

The following is an example of querying filters for all data sources:

```
? FILTER
```

If no filters are defined, the following message displays

```
NO FILTERS DEFINED
```

If filters are defined, the following screen displays:

```
Set File      Filter name Description
-----
      CAR      ROB      Rob's selections
*   CAR      PETER      Peter's selections for CAR
*   EMPLOYEE DAVE      Dave's tests
      EMPLOYEE BRAD      Brad's tests
```

To query filters for the CAR data source, issue:

```
? FILTER CAR
```

If no filters are defined for the CAR data source, the following message displays

```
NO FILTERS DEFINED FOR FILE NAMED CAR
```

If filters are defined for the CAR data source, the following screen displays:

```
Set File      Filter name Description
-----
      CAR      ROB      Rob's selections
*   CAR      PETER      Peter's selections for CAR
```

To see all active filters, issue the following command

```
? FILTER * SET
```

The output is:

```
Set File      Filter name Description
-----
*   CAR      PETER      Peter's selections for CAR
*   EMPLOYEE DAVE      Dave's tests
```

The asterisk in the first column indicates that a filter is activated.

Applying Filters to Joined Structures

When you report against a joined structure using a host file name, none of the original filters are active. Once the join is in effect, if you declare filters for the host file name, they apply to the joined structure and remain in effect as long as the join is in effect. When the join is cleared, the original filters for the host file are reactivated.

If you use the cross-referenced file name in a request, its activated filters are implemented. For related information see Chapter 13, *Joining Data Sources*.

```

*****
-* JOIN AND FILTER INTERACTION
*****

-* DECLARE A FILTER
FILTER FILE EMPLOYEE CLEAR
    NAME=XXX WHERE JOBCODE EQ 'A01'
END
SET FILTER = XXX IN EMPLOYEE ON
-* EMPLOYEE FILE SHOWS JOBCODE A01 ONLY
TABLE FILE EMPLOYEE PRINT EMP_ID JOBCODE
END
-* -----
-* NOW JOIN TO JOBFILE AND REDECLARE THE SAME FILTER WITH A DIFFERENT
JOBCODE
-* -----
JOIN JOBCODE IN EMPLOYEE TO JOBCODE IN JOBFILE
FILTER FILE EMPLOYEE
    NAME=XXX WHERE JOBCODE EQ 'A07'
END
-* (NOTE: NEW FILTER FOR JOIN STRUCTURE IS NOT ACTIVATED YET)
-* EMPLOYEE FILE SHOWS **ALL** JOBCODES (ORIGINAL FILTER TURNED OFF BY
JOIN)
TABLE FILE EMPLOYEE PRINT EMP_ID JOBCODE
END
-* -----
-* NOW TURN ON THE NEW FILTER THAT APPLIES TO THE JOIN STRUCTURE
-* -----
SET FILTER = XXX IN EMPLOYEE ON
-* SHOWS JOBCODE A07 (NOT A01) (NEW FILTER APPLIES TO JOIN ONLY)
TABLE FILE EMPLOYEE PRINT EMP_ID JOBCODE
END
-* NOW CLEAR THE JOIN TO RE-ESTABLISH THE ORIGINAL FILTER
JOIN CLEAR *
-* NOW SHOWS JOBCODE A01 ONLY, AS BEFORE (ORIGINAL FILTER REACTIVATED)
TABLE FILE EMPLOYEE PRINT EMP_ID JOBCODE
END

```

VSAM Record Selection Efficiencies

The most efficient way to retrieve selected records from a VSAM KSDS data source is by applying an IF screening test against the primary key. This results in a direct read of the data using the data source's index. Only those records that you request are retrieved from the file. The alternative method of retrieval, the sequential read, forces the data adapter to retrieve all the records into storage.

Selection criteria that are based on the entire primary key, or on a subset of the primary key, cause direct reads using the index. A partial key is any contiguous part of the primary key beginning with the first byte.

IF selection tests performed against virtual fields can take advantage of these efficiencies as well, if the full or partial key is embedded in the virtual field.

The EQ and IS relations realize the greatest performance improvement over sequential reads. When testing on a partial key, they retrieve only the first segment instance of the screening value. To retrieve subsequent instances, NEXT logic is used.

Screening relations GE, FROM, FROM-TO, GT, EXCEEDS, IS-MORE-THAN, and NOT-FROM-TO all obtain some benefit from direct reads. The following example uses the index to find the record containing primary key value 66:

```
IF keyfield GE 66
```

Then, it continues to retrieve records by sequential processing because VSAM stores records in ascending key sequence. The direct read is not attempted when the IF screening conditions NE, IS-NOT, CONTAINS, OMITS, LT, IS-LESS-THAN, LE, and NOT-FROM are used in the report request.

Reporting From Files With Alternate Indexes

Similar performance improvement is available for ESDS and KSDS files that use alternate indexes. An alternate index provides access to records in a key sequenced data set based on a key other than the primary key.

All benefits and limitations inherent with screening on the primary or partial key are applicable to screening on the alternate index or partial alternate index. For a discussion of these efficiencies, refer to *VSAM Record Selection Efficiencies* on page 5-39.

Note: It is not necessary to take an explicit indexed view to use the index.

CHAPTER 6

Creating Temporary Fields

Topics:

- The Difference Between DEFINE and COMPUTE
- Defining a Virtual Field
- Computing Calculated Values
- Calculating Trend Values and Forecasts
- Using Functions With Temporary Fields
- Creating Temporary Fields Unrelated to Master Files

Frequently, a report requires information that does not exist in a data source, but can be derived from data source fields. You can derive this information by creating a temporary field.

A temporary field does not take up any storage space in the data source; it is created only when needed. The value of a temporary field is the result of an expression. An expression combines fields, constants, and operators to produce a single value. You can specify the operations yourself, or you can use the many supplied functions to perform specific calculations or manipulations. In addition, you can use expressions and functions as building blocks for more complex expressions, as well as use one temporary field to evaluate another.

You can create the following types of temporary fields:

- Virtual fields, by issuing a DEFINE command or by including a DEFINE attribute in a Master File. See *Defining a Virtual Field* on page 6-4.
- Calculated values, by issuing a COMPUTE command. See *Computing Calculated Values* on page 6-11.
- Virtual fields that are independent of a Master File using a DEFINE function. See *Creating Temporary Fields Unrelated to Master Files* on page 6-33.

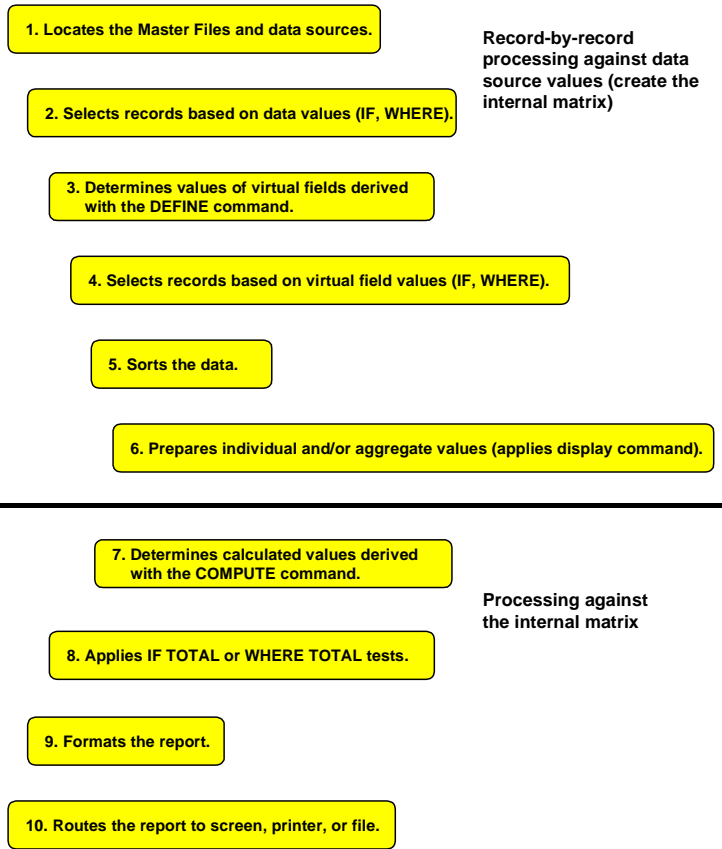
In addition to being used for reporting, temporary fields can be created for other applications, such as Dialogue Manager stored procedures. See the *Developing Applications* manual for details.

The Difference Between DEFINE and COMPUTE

The major difference between DEFINE and COMPUTE is the point of execution:

- A virtual field (DEFINE) is evaluated as each record that meets the selection criteria is retrieved from the data source. The result of the expression is treated as though it were a real field stored in the data source. A virtual field is in effect until it is cleared or the session ends.
- A calculated value (COMPUTE), works on the results of a SUM, PRINT, or COUNT command for a specific report request. The value is calculated after all the data that meets the selection criteria is retrieved, sorted, and summed. The calculation, therefore, is performed using the aggregated values of the fields referenced in the COMPUTE command.

The following diagram illustrates when and how a request processes DEFINE and COMPUTE commands:



Example

Distinguishing Between DEFINE and COMPUTE

The difference between DEFINE and COMPUTE is best illustrated using a SUM command in the request:

```
DEFINE FILE SALES
DRATIO = DELIVER_AMT/OPENING_AMT;
END

TABLE FILE SALES
SUM DELIVER_AMT AND OPENING_AMT AND DRATIO
COMPUTE CRATIO = DELIVER_AMT/OPENING_AMT;
END
```

The same expression is used to evaluate both the virtual field, DRATIO, and the calculated value, CRATIO, but their results are very different:

DELIVER_AMT	OPENING_AMT	DRATIO	CRATIO
760	724	28.41	1.05

The value for CRATIO is calculated after all records have been selected, sorted, and aggregated. The calculation is performed using the aggregated values of the fields that it references. The virtual field DRATIO is calculated on each retrieved record.

Reference

Number of Temporary Fields Allowed

Temporary fields created by either the DEFINE or COMPUTE command are dependent on the amount of memory available and are counted against the maximum number of display fields. Their command syntax accepts the same types of valid expressions. For details on determining the maximum number of display fields that can be used in a request, see Chapter 1, *Creating Tabular Reports*.

Defining a Virtual Field

The DEFINE command creates virtual fields that may then be used in a request as though they were real data source fields.

The calculation that determines the value of a virtual field is performed on each retrieved record that passes any screening conditions on real fields. The result of the expression is treated as though it were a real field stored in the data source.

The virtual field is in effect for the duration of the session in which it is issued, unless cleared by:

- A DEFINE FILE *filename* CLEAR command.
- A subsequent DEFINE command—without the ADD phrase—against the same data source.
- A JOIN command.

In addition to using the DEFINE command, you can declare virtual fields in a Master File. These virtual fields are available whenever the data source is used for reporting. The FIELDNAME SET parameter determines whether virtual fields in the Master File can refer to fields located in different segments. Fields created with the DEFINE command can refer to virtual fields in the Master File. Unlike fields created with the DEFINE command, virtual fields in the Master File are not cleared by JOIN or DEFINE FILE *filename* CLEAR commands.

Syntax

How to Create a Virtual Field

Specify the DEFINE command before you begin a report request.

```
DEFINE FILE filename[.view_fieldname] [CLEAR|ADD]  
  
fieldname[/format]=expression;  
fieldname[/format][WITH realfield]=expression;  
fieldname[/format] REDEFINES qualifier.fieldname=expression;  
.  
.  
.  
END
```

where:

filename

Is the name of the data source for which you are defining the virtual field.

If the report request specifies an alternate view, use *filename* in conjunction with *view_fieldname*.

Note that all of the fields used to define the virtual field must lie on a single path in the data source. If they do not, you can use an alternate view, which requires alternate view DEFINE commands. For an alternate view, virtual fields cannot have qualified field names or field names that exceed the 12-character limit. For information on alternate views, see Chapter 15, *Improving Report Processing*.

view_fieldname

Is the field on which an alternate view is based in the corresponding request. You may need to use an alternate view if the fields used do not lie on a single path in the normal view.

CLEAR

Clears previously defined virtual fields associated with the specified data source. This value is the default.

ADD

Enables you to specify additional virtual fields for a data source without releasing any existing virtual fields. Omitting ADD produces the same results as the CLEAR option.

fieldname

Is a name of up to 66 characters. Indexed field names must be less than or equal to 12 characters. It can be the name of a new virtual field that you are defining or an existing field declared in the Master File, which you want to redefine.

The name can include any combination of letters, digits, and underscores (_) and should begin with a letter.

Do not use field names of the type *Cn*, *En*, or *Xn* (where *n* is any sequence of one or two digits) because they are reserved for other uses.

format

Is the format of the field. All formats except text fields (TX) are allowed. The default value is D12.2. For information on field formats, see the *Describing Data* manual.

WITH realfield

Associates a virtual field with a data source segment containing a real field. For more information see *Usage Notes for Creating Virtual Fields* on page 6-6.

REDEFINES qualifier.fieldname

Enables you to redefine or recompute a field whose name exists in more than one segment.

expression

Can be an arithmetic or logical expression or function, evaluated to establish the value of *fieldname* (see Chapter 8, *Using Expressions*). You must end each expression with a semicolon except the last one, where the semicolon is optional.

Fields in the expression can be real data fields, data fields in data sources that are cross-referenced or joined, or previously defined virtual fields. For related information see *Usage Notes for Creating Virtual Fields* on page 6-6.

END

Is required to end the DEFINE FILE command.

Reference

Usage Notes for Creating Virtual Fields

- When a JOIN is issued for a data source, all pre-existing virtual fields for that data source are cleared except those defined in the Master File. This may affect virtual fields used in an expression.
- To join structures using a virtual field, make sure the DEFINE follows the JOIN command. See Chapter 13, *Joining Data Sources*, for an explanation of reporting on joined data sources.
- Virtual fields declared in a Master File are not affected by the DEFINE FILE *filename* CLEAR command.
- If no field in the expression is in the Master File or has been defined, use the WITH command to identify the logical home of the defined calculation. See *Establishing a Segment Location for a Virtual Field* on page 6-8.
- WITH can be used to move the logical home for the virtual field to a lower segment than it would otherwise be assigned to (for example, to count instances in a lower segment).
- You may define fields simultaneously (in addition to fields defined in the Master File) for as many data sources as desired. The total length of all virtual fields and real fields cannot exceed 16,000 characters. This limit may be increased at installation time. Please refer to your installation guide. For WebFOCUS the total length of all virtual fields and real fields cannot exceed 32,000 characters.
- When you specify virtual fields in a request, they count toward the display field limit. For details on determining the maximum number of display fields that can be used in a request, see Chapter 1, *Creating Tabular Reports*.
- Virtual fields are only available when the data source is used for reporting. Virtual fields cannot be used with MODIFY.
- A DEFINE may not contain qualified field names on the left-hand side of the expression. If the same field name exists in more than one segment, and that field must be redefined or recomputed, use the REDEFINES command.
- When the value of SET FIELDNAME is changed, virtual fields are cleared. For details see the *Developing Applications* manual.
- Using a self-referencing DEFINE such as $x=x+1$ disables AUTOPATH (see the *Developing Applications* manual).

Example**Defining a Virtual Field**

In the following request, the value of **RATIO** is calculated by dividing the value of **DELIVER_AMT** by **OPENING_AMT**. The **DEFINE** command creates **RATIO** as a virtual field, which is used in the request as though it were a real field in the data source:

```
DEFINE FILE SALES
RATIO = DELIVER_AMT/OPENING_AMT;
END

TABLE FILE SALES
PRINT DELIVER_AMT AND OPENING_AMT AND RATIO
WHERE DELIVER_AMT GT 50
END
```

The output is:

PAGE 1

DELIVER_AMT	OPENING_AMT	RATIO
80	65	1.23
100	100	1.00
80	90	.89

Defining Multiple Virtual Fields

You may wish to have more than one **DEFINE** command referring to the same data source and to use some or all of the virtual fields in the request. The **ADD** option enables you to specify additional virtual fields without clearing existing ones. If you omit the **ADD** option, previously defined virtual fields in that data source are cleared.

If you want to clear a virtual field for a particular data source, use the **CLEAR** option.

ADD and **CLEAR** options are useful when you are executing requests from stored procedures or are creating interactive stored procedures (explained in the *Developing Applications* manual). To see which virtual fields are available, use the **? DEFINE** command described in the *Developing Applications* manual.

Example

Adding and Clearing Virtual Fields

The following annotated example illustrates the use of the ADD and CLEAR options for virtual fields:

1.

```
DEFINE FILE CAR
  ETYPE/A2=DECODE STANDARD (OHV O OHC O ELSE L);
  END
```
2.

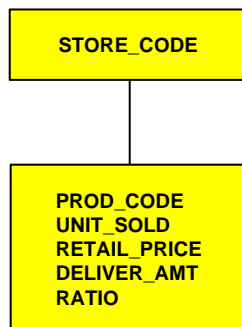
```
DEFINE FILE CAR ADD
  TAX/D8.2=IF MPG LT 15 THEN .06*RCOST
    ELSE .04*RCOST;
  FCOST = RCOST+TAX;
  END
```
3.

```
DEFINE FILE CAR CLEAR
  COST = RCOST-DCOST;
  END
```

1. The first DEFINE command creates the ETYPE virtual field for the CAR data source. For information about the DECODE function, see the *Developing Applications* manual.
2. Two more virtual fields, TAX and FCOST, are created for the CAR data source. The ADD option allows you to reference ETYPE, TAX, and FCOST in future requests.
3. The CLEAR option clears the three previously defined virtual fields and only the COST virtual field in the last DEFINE is available for further requests.

Establishing a Segment Location for a Virtual Field

Virtual fields have a logical location in the data source structure just like permanent data source fields. The logical home of a virtual field is on the lowest segment that has to be accessed in order to evaluate the expression. The logical home of a virtual field determines its time of execution. Consider the following data source structure and the following DEFINE command:



```
RATIO = DELIVER_AMT/RETAIL_PRICE ;
```

The expression for RATIO includes at least one real data source field. As far as report capabilities are concerned, the field RATIO is just like a real field in the Master File and is located in the lowest segment.

In some applications, you can have a virtual field evaluated by an expression that contains no real data source fields. Such an expression might refer to only temporary fields or literals. For example,

```
NCOUNT/I5 = NCOUNT+1;
```

or

```
DATE/YMD = '19990101';
```

Since neither expression contains a data source field (NCOUNT and the literal do not exist in the Master File), their logical position in the data source cannot be determined. You have to specify in which segment you want them to be placed. To associate a virtual field with a specific segment, use the WITH phrase. The field name following WITH may be any real field in the Master File.

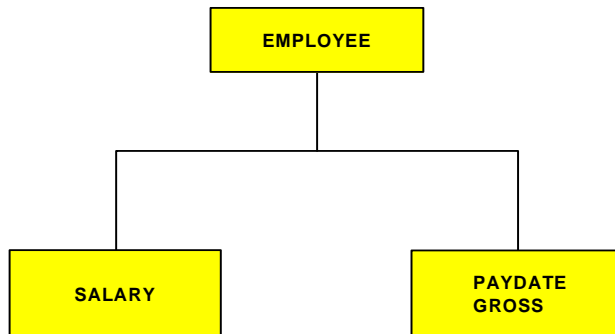
For example, the field NCOUNT is placed in the same segment as UNITS. NCOUNT is calculated each time a new segment instance is retrieved.

```
DEFINE FILE GGSales
NCOUNT/I5 WITH UNITS = NCOUNT+1;
END
```

You may be able to increase the retrieval speed with an external index on the virtual field. In this case, you can associate the index with a target segment outside of the segment containing the virtual field. See the *Developing Applications* manual for more information on external indexes.

Defining Virtual Fields Using a Multi-Path Data Source

The expression of a virtual field may include fields from all segments of a data source, but they must lie in a unique top-to-bottom path. Different virtual fields may, of course, lie along different paths. For example, consider the following data source structure:



This data source structure would not permit you to write the following expression:

```
NEWAMT = SALARY+GROSS;
```

The expression is invalid because the structure implies that there can be several SALARY segments for a given EMPLOYEE and it is not clear which SALARY to associate with which GROSS.

To accomplish such an operation, you can use the alternate view option explained in Chapter 15, *Improving Report Processing*.

Increasing the Speed of DEFINE Calculations

Calculations for DEFINE are fully compiled, by default, into machine code when the request is parsed. The machine code is then executed to perform the calculations at run time. This increases the speed and efficiency of your calculations.

Certain calculations are ineligible for compilation:

- Calculations that involve any function (for example, user functions), except for EDIT, DECODE, and LAST.
- Calculations that test for existing data (IF field IS-NOT MISSING) or that result in a missing field (TEMP/A4 MISSING ON= ...).
- Calculations that involve fields with date formats. (See the table of date formats in the FORMAT attribute description in the *Describing Data* manual.)
- Calculations that use exponentiation ($10^{**}2$).

In earlier product releases, calculations for DEFINE were converted to an internal form and interpreted at execution time. The ineligible calculations listed above are automatically processed using this conversion logic.

To prevent compilation into machine code, apply the following SET option before issuing a DEFINE command:

```
SET COMPUTE = OLD
```

Calculations are processed by the conversion logic until the session ends or the SET option is changed back to NEW. OLD requires less memory but takes longer to execute. See the *Developing Applications* manual for information about SET COMPUTE.

Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN

Occasionally, new code needs to be added to an existing application. When adding code, there is always the possibility of over-writing existing virtual fields by reusing their names inadvertently.

The DEFINE FILE SAVE command forms a new context for virtual fields, which can then be removed with DEFINE FILE RETURN. Each new context creates a new layer or command environment. When you first enter the new environment, all of the virtual fields defined in the previous layer are available in the new layer. Overwriting or clearing a virtual field definition affects only the current layer. When you return to the previous layer using DEFINE FILE RETURN, its virtual field definitions are intact.

Therefore, all the virtual fields that are created in the new application can be removed before returning to the calling application, without affecting existing virtual fields in that application.

For an example of DEFINE FILE SAVE and DEFINE FILE RETURN, see Chapter 13, *Joining Data Sources*.

Note: A JOIN command can be issued after a DEFINE FILE SAVE command. However, in order to clear the join context, you must issue a JOIN CLEAR command if the join is still in effect. If only virtual fields and DEFINE FILE ADD were issued after a DEFINE FILE SAVE command, you can clear them by issuing a DEFINE FILE RETURN command.

Syntax

How to Protect Virtual Fields From Being Over-Written

```
DEFINE FILE filename SAVE  
fld1/format1=expression1 ...  
fld2/format2=expression2 ...  
END...  
TABLE FILE filename ...  
MODIFY FILE filename ...  
DEFINE FILE filename RETURN
```

where:

SAVE

Creates a new context for virtual fields.

filename

Is the name of the Master File that gets a new context and has the subsequent virtual fields applied before the DEFINE FILE RETURN command is issued.

RETURN

Clears the current context if it was created by DEFINE FILE SAVE and restores the previous context.

Computing Calculated Values

The COMPUTE command calculates one or more temporary fields in the request. Values calculated by the COMPUTE command are available for only the specified report request.

A calculated value works on the results of a SUM, PRINT, or COUNT command, and is calculated after all records have been selected, sorted, and summed. The column calculation, therefore, is performed using the summed field values of the fields that it references.

Syntax

How to Calculate a Field Value

You specify the COMPUTE command in the body of the report request, following the display command, and optionally introduced by AND. You can compute more than one field with a single COMPUTE command. Following are a number of syntax variations that can be used separately or in combination.

To compute one or more new columns:

```
COMPUTE fld1 [/format]= expression;[AS 'title']
COMPUTE fld2 [/format]= expression;[AS 'title']
```

To suppress the display of a calculated column:

```
COMPUTE fld1 [/format]= expression; NOPRINT
COMPUTE fld2 [/format]= expression; NOPRINT
```

To position the calculated column:

```
COMPUTE fld1 [/format]= expression;[IN [+]n]
COMPUTE fld2 [/format]= expression;[IN [+]n]
```

To allocate a column without a calculation:

```
COMPUTE fld1 [/format]= ;
           fld2 [/format]= ;
```

where:

fld[1...2...]

Is the name of the calculated value.

The name can be up to 66 characters long and can include any combination of letters, digits, and underscores (_), and should begin with a letter. Other characters are not recommended and may cause problems in some operating environments or when resolving expressions.

Do not use field names of the type *Cn*, *En*, and *Xn* (where *n* is any sequence of one or two digits) because they are reserved for other uses.

format

Is the format of the field. All formats except text fields (TX) are allowed. The default is D12.2. For information on formats, see the *Describing Data* manual.

expression

Can be an arithmetic and/or logical expression or function (see Chapter 8, *Using Expressions*). Each field used in the expression must be part of the request. Each expression must end with a semicolon.

NOPRINT

Suppresses the printing of the field. For more information, see Chapter 9, *Customizing Tabular Reports*.

AS 'title'

Changes the name of the calculated value. For more information, see in Chapter 9, *Customizing Tabular Reports*.

IN

Specifies the location of the column. For more information, see Chapter 9, *Customizing Tabular Reports*.

Reference**Usage Notes for Calculated Field Values**

- If you specify any optional COMPUTE phrases (such as, AS, IN, or NORPINT), and you compute additional fields following these phrases, you must repeat the commands COMPUTE or AND COMPUTE before specifying the additional fields.
- You can rename and justify column totals and row totals. See the examples in Chapter 7, *Including Totals and Subtotals*.
- Expressions in a COMPUTE command can include fields with prefix operators (see Chapter 2, *Displaying Report Data*). For more information on valid expressions, see Chapter 8, *Using Expressions*.
- Fields referred to in a COMPUTE command are counted toward the display field limit and appear in the internal matrix, unless they have been previously referenced. For details on determining the maximum number of display fields that can be used in a request, see Chapter 1, *Creating Tabular Reports*.

Example**Calculating a Field Value**

In the following example, the COMPUTE command creates a temporary field REVENUE based on the product of UNIT_SOLD and RETAIL_PRICE and displays information for New York City. The format D12.2M indicates the field format for REVENUE and the AS command changes the default column headings for UNIT_SOLD and RETAIL_PRICE. REVENUE is only available for this report request.

```
TABLE FILE SALES
HEADING CENTER
"NEW YORK PROFIT REPORT"
" "
SUM UNIT_SOLD AS 'UNITS,SOLD' RETAIL_PRICE AS 'RETAIL,PRICE'
COMPUTE REVENUE/D12.2M = UNIT_SOLD * RETAIL_PRICE;
BY PROD_CODE AS 'PROD,CODE'
WHERE CITY EQ 'NEW YORK'
END
```

The output is:

```
NEW YORK PROFIT REPORT
```

PROD CODE	UNITS SOLD	RETAIL PRICE	REVENUE
----	-----	-----	-----
B10	30	\$.85	\$25.50
B17	20	\$1.89	\$37.80
B20	15	\$1.99	\$29.85
C17	12	\$2.09	\$25.08
D12	20	\$2.09	\$41.80

Using Positional Column Referencing With Calculated Values

In a COMPUTE command, it is sometimes convenient to refer to a field by its report column position rather than its name. This option is especially useful when the same field is specified for several report columns.

Column referencing becomes essential when using the same field name in a variety of ways. The columns produced by display commands (whether displayed or not) can be referred to as C1 for the first column, C2 for the second column, and so forth. The BY field columns are not counted.

Example

Using Positional Column Referencing

The following example demonstrates positional field references in a COMPUTE:

```
TABLE FILE CAR
SUM AVE.DEALER_COST
SUM AVE.DEALER_COST AND COMPUTE RATIO=C1/C2;
BY COUNTRY
END
```

The columns produced by display commands can be referred to as C1 for the first column (AVE.DEALER_COST), C2 for the second column (AVE.DEALER_COST BY COUNTRY), and so forth. The BY field columns are not counted.

The output is:

PAGE 1

AVE DEALER_COST	COUNTRY	AVE DEALER_COST	RATIO
7,989	ENGLAND	9,463	.84
	FRANCE	4,631	1.73
	ITALY	10,309	.77
	JAPAN	2,756	2.90
	W GERMANY	7,795	1.02

Using COMPUTE and ACROSS

If the COMPUTE command is part of a display command, a new column is calculated for each set of field values.

If the COMPUTE command is issued right after an ACROSS phrase, only a recap type of the calculation is performed once for all columns.

Example

Using Compute as Part of a Display Command

```
TABLE FILE SALES
SUM UNIT_SOLD COMPUTE NEWVAL = UNIT_SOLD * RETAIL_PRICE;
ACROSS CITY
END
```

The first page of output is:

PAGE 1

CITY							
NEW YORK		NEWARK		STAMFORD		UNIONDALE	
UNIT_SOLD	NEWVAL	UNIT_SOLD	NEWVAL	UNIT_SOLD	NEWVAL	UNIT_SOLD	NEWVAL
162	1,764.18	42	104.16	376	4,805.28	65	297.70

Example

Using COMPUTE After an ACROSS Phrase

```
TABLE FILE SALES
SUM UNIT_SOLD AND RETURNS
WHERE DATE GE '010' AND DATE LE '1031'
ACROSS DATE
COMPUTE
TOT_UNITS/D5=C1 + C3 + C5;
TOT_RETURNS = C2 + C4 + C6;
END
```

C1, C2, C3, C4, C5, and C6 are positional column references.

The output is:

PAGE 1

DATE							
10/17		10/18		10/19		TOT_UNITS	TOT_RETURNS
UNIT_SOLD	RETURNS	UNIT_SOLD	RETURNS	UNIT_SOLD	RETURNS		
162	15	78	2	29	1	269	18.00

Sorting Calculated Values

You can sort a report by either a virtual field or a calculated value. To sort by a calculated value, you must use the BY TOTAL phrase in your request. For details see Chapter 4, *Sorting Tabular Reports*.

Screening on Calculated Values

You can screen on values produced by COMPUTE commands by using the WHERE TOTAL test, as described in Chapter 5, *Selecting Records for Your Report*.

Example

Screening on Calculated Values

The following example illustrates how to screen on values produced by a COMPUTE command. The selection is performed using WHERE TOTAL, a requirement for screening on a calculated value.

```
TABLE FILE SALES
PRINT UNIT_SOLD
AND COMPUTE NEWSALES = UNIT_SOLD * 1.1;
BY CITY
WHERE TOTAL NEWSALES GT 10
END
```

The output is:

PAGE 1

CITY	UNIT_SOLD	NEWSALES
NEW YORK	30	33.00
	20	22.00
	15	16.50
	12	13.20
	20	22.00
	30	33.00
	35	38.50
NEWARK	29	31.90
	13	14.30
STAMFORD	60	66.00
	40	44.00
	29	31.90
	25	27.50
	45	49.50
	27	29.70
	80	88.00
	70	77.00
UNIONDALE	25	27.50
	40	44.00

Calculating Trend Values and Forecasts

The FORECAST feature allows you to uncover trends in numeric data. Depending on the options you specify, it can also provide predicted values beyond the range of the values stored in the data source.

The methods available for calculating trend values are:

- Simple moving average (MOVAVE). This method calculates a series of arithmetic means using a user-specified number of values from a report column. For details, see *Using a Simple Moving Average* on page 6-21.
- Exponential moving average (EXPAVE). This method calculates a weighted average between the previously calculated value of the average and the next data point. For details, see *Using an Exponential Moving Average* on page 6-25.
- Linear regression analysis (REGRESS). This method derives the coefficients of a straight line that best fits the data points and uses this linear equation to estimate values. For details, see *Using a Linear Regression Equation* on page 6-27.

To generate predicted values, FORECAST continues the same calculations beyond the data points by using the generated trend values as new data points. For the REGRESS technique, the calculated regression equation is used to derive trend and predicted values.

FORECAST Processing

You invoke FORECAST using a special version of the ON *sortfield* RECAP command. In this command you specify the parameters needed for generating estimated values, including the field to be used in the calculations, the method to use, and the number of predictions to generate. The RECAP field can be a new field or it can be the same field used in the FORECAST calculations:

- If the RECAP field is the same as the field being used to generate the FORECAST calculations, it is referred to as a *recursive* FORECAST. In this case, the original field is not printed, even if it was referenced in the display command, and the RECAP column contains the original field values followed by the number of predicted values specified in the FORECAST syntax. No trend values display in the report. However, the original column will be propagated to an output file unless you set HOLDLIST to PRINTONLY
- If the RECAP field is a new field, the original field and the new field both display in the report output (if the original field was mentioned in the display command). This is referred to as a *non-recursive* FORECAST. The new field will contain trend values (estimated values *within* the range of the existing data points) and, depending on the arguments you supply, forecast values (predictions *beyond* the range of the existing data points).

The sort field used for FORECAST must be a numeric or smart date field. FORECAST operates on the last ACROSS field in the request. If the request contains no ACROSS fields, it operates on the last BY field. However, to use an ACROSS field with FORECAST the display command must be SUM (or its synonyms, ADD or WRITE) or COUNT. The display command cannot be PRINT or LIST. The FORECAST calculations start over when the highest-level sort field changes its value. In a request with multiple display commands, FORECAST operates on the last ACROSS field (or if there are no ACROSS fields, the last BY field) of the last display command.

Although you pass parameters to FORECAST using an argument list in parentheses, FORECAST is not a function. It can coexist with a user-written subroutine of the same name, as long as the user-written subroutine is not specified in a RECAP command.

Syntax

How to Use FORECAST

The following syntax is for the MOVAVE and EXPAVE methods:

```
ON sfld RECAP fld1[/fmt] = FORECAST(fld2, interval, npredict,  
'method',npoint);
```

The following syntax is for the REGRESS method (omits the *npoint* parameter):

```
ON sfld RECAP fld1[/fmt] = FORECAST(fld2, interval, npredict, 'REGRESS');
```

where:

sfld

Is the last ACROSS field in the request and must be a numeric or smart date field. If the request contains no ACROSS phrases, FORECAST works on the last BY field. However, FORECAST is only supported with ACROSS when the display command is SUM, WRITE, ADD, or COUNT.

fld1

Is a numeric field. It can be a real field, a virtual field, or a calculated field.

Note: The word FORECAST and the opening parenthesis must be on the same line as the syntax *fld1*=.

fmt

Is the display format for *fld1*. If it is omitted, the default format is D12.2. Even if *fld1* was previously reformatted using a DEFINE or COMPUTE command, the format specified in the RECAP command is respected.

fld2

Is any numeric field. If it is the same as *fld1* (recursive), the predicted values will be appended to the report column after the data values. The original column is not printed in the report. If it is a different name than *fld1* (non-recursive), this new column will be calculated containing both trend values (estimated values within the range of the existing data points) and, if you specify a non-zero number of predictions, forecast values (predictions beyond the range of the existing data points), while retaining the original field as a separate report column, if it was referenced in the display command.

interval

Is the increment to add to each *sfl*d value (after the last data point) to get to the next. It must be a positive whole number. To sort in descending order, use the BY HIGHEST phrase. The result of adding this number to the *sfl*d values will be converted to the same format as *sfl*d.

For date fields, the minimal component in the format determines how the number is interpreted. For example, if the format is YMD, MDY, or DMY, an interval value of 2 is interpreted as meaning two days; if the format is YM, the 2 is interpreted as meaning two months.

npredict

Is the number of predictions for FORECAST to calculate. It must be a whole number greater than or equal to zero. Zero indicates that you want no predictions. Zero is only supported when the RECAP field is a new field (non-recursive), not the same field used as the first parameter to FORECAST. If you supply a number that is not a whole number, the fractional portion is dropped.

method

Is the estimation method to use. It can be one of the following values enclosed in single quotation marks:

Method	Definition
MOVAVE	Simple moving average
EXPAVE	Exponentially smoothed moving average
REGRESS	Linear regression

npoint

Is a positive whole number that specifies the number of values to average for the MOVAVE method. For EXPAVE, this number is used to calculate the weights for each component in the average. This parameter must be specified for MOVAVE and EXPAVE and omitted for REGRESS. If you supply a number that is not a whole number, the fractional portion is dropped.

Reference

Usage Notes for FORECAST

- For averages, data values should be evenly spaced in order to get meaningful results.
- The RECAP command used with FORECAST can contain only the FORECAST syntax. FORECAST does not recognize any syntax after the closing semicolon (;). To specify options such as AS or IN:
 - If it is a non-recursive FORECAST request (creates a new field), use an empty COMPUTE command prior to the RECAP.
 - If it is a recursive FORECAST request, specify the options when the field is first referenced in the report request.
- FORECAST operates on the last ACROSS field, and if the request contains no ACROSS phrases, FORECAST operates on the last BY field.
- FORECAST is only supported for ACROSS fields if the display command is SUM, COUNT, WRITE, or ADD. You cannot use FORECAST on an ACROSS field if the display command is LIST or PRINT.
- In a request with multiple display commands, FORECAST must be applied to the last ACROSS field (or if there are no ACROSS fields, the last BY field) in the last display command. If you use FORECAST to recalculate a field in the request, the original value of the field will be used everywhere except in the columns displayed by the last display command in the request.
- BY TOTAL is not supported.
- MORE, MATCH, FOR, and OVER are not supported.
- The LINES and RECORDS statistics are affected by FORECAST.
- The process of generating the FORECAST values creates extra columns that are not printed in the report output. The number and placement of these additional columns varies depending on the specific request. Therefore, use of column notation is not supported in a request that includes FORECAST.
- SUMMARIZE and RECOMPUTE are not supported for the same sort field used for FORECAST.
- FORECAST is not supported for the FOCUS GRAPH facility; it is supported for the iWay GRAPH facility.
- A request can contain up to seven non-FORECAST RECAP commands and up to seven additional FORECAST RECAP commands.
- The left side of a RECAP command used for FORECAST supports the CURR attribute for currency conversions. The MISSING attribute is not supported in the RECAP command.
- A request that creates a new field using the REGRESS method with the SUM command produces a regression based on the detail data values, not the summed values. This is not true if the REGRESS method is used to replace an existing field.

Forecasting Methods

The methods available with FORECAST may sometimes be used to predict values outside the range of the existing data points. However, these methods are not always reliable predictors. Many factors determine how accurate a prediction will be. The FORECAST operation performs the calculations based on the data provided. Decisions about their use and reliability are the user's responsibility.

Using a Simple Moving Average

A simple moving average is a series of arithmetic means calculated with a user-specified number of values, n , from a report column. Each new mean in the series is calculated by dropping the first value used in the prior calculation and adding the next data value to the calculation.

Simple moving averages are sometimes used to analyze trends in stock prices over time. In this scenario, the average is calculated using n periods worth of stock prices. A disadvantage to this indicator is that because it drops the oldest values from the calculation as it moves on in time, it loses its memory over time. Also, mean values are distorted by extreme highs and lows and give equal weight to each point.

Predicted values beyond the range of the data values are calculated using a moving average that treats the calculated trend values as new data points.

The first complete moving average occurs at the n th data point because the calculation requires n values. This is called the lag. The moving average values for the lag rows are calculated as follows: the first value in the moving average column is equal to the first data value, the second value in the moving average column is the average of the first two data values, and so on until the n th row at which point there are enough values to calculate the moving average with the number of values specified.

Example Calculating a New Simple Moving Average Column

This request defines an integer value named PERIOD to use as the independent variable for the moving average. It predicts three periods of values beyond the range of the retrieved data.

```
DEFINE FILE GGSALES
  SDATE/YYM = DATE;
  SYEAR/Y = SDATE;
  SMONTH/M = SDATE;
  PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
  SUM UNITS DOLLARS
  BY CATEGORY BY PERIOD
  WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
  ON PERIOD RECAP MOVAVE/D10.1= FORECAST(DOLLARS,1,3,'MOVAVE',3);
END
```

The output is:

Category	PERIOD	Unit Sales	Dollar Sales	MOVAVE
-----	-----	-----	-----	-----
Coffee	1	61666	801123	801,123.0
	2	54870	682340	741,731.5
	3	61608	765078	749,513.7
	4	57050	691274	712,897.3
	5	59229	720444	725,598.7
	6	58466	742457	718,058.3
	7	60771	747253	736,718.0
	8	54633	655896	715,202.0
	9	57829	730327	711,158.7
	10	57012	724412	703,545.0
	11	51110	620264	691,667.7
	12	58981	762328	702,334.7
	13	0	0	694,975.6
	14	0	0	719,879.4
	15	0	0	705,729.9
Food	1	54394	672727	672,727.0
	2	54894	699073	685,900.0
	3	52713	642802	671,534.0
	4	58026	718514	686,796.3
	5	53289	660740	674,018.7
	6	58742	734705	704,653.0
	7	60127	760586	718,677.0
	8	55622	695235	730,175.3
	9	55787	683140	712,987.0
	10	57340	713768	697,381.0
	11	57459	710139	702,349.0
	12	57290	705315	709,740.7
	13	0	0	708,398.2
	14	0	0	707,818.0
	15	0	0	708,652.3

Note:

- The number of values to use in the average is 3.
- Three predicted values of MOVAVE are calculated within each value of CATEGORY. For values outside the range of the data, new PERIOD values are generated by adding the interval value (1) to the prior PERIOD value
- There are no UNITS or DOLLARS values for the generated PERIOD values.
- Each average (MOVAVE value) is computed using DOLLARS values where they exist. For predicted values beyond those points, the calculated MOVAVE values are used as new data points to continue the moving average, PERIOD is the independent variable (x) and MOVAVE is the dependent variable (y).

The first MOVAVE value (801,123.0) is equal to the first DOLLARS value.

The second MOVAVE value (741,731.5) is the mean of DOLLARS values one and two: $(801123 + 682340) / 2$.

The third MOVAVE value (749,513.7) is the mean of DOLLARS values one through three: $(801123 + 682340 + 765078) / 3$.

The fourth MOVAVE value (712,897.3) is the mean of DOLLARS values two through four: $(682340 + 765078 + 691274) / 3$.

The predicted MOVAVE values (starting with 694,975.6 for PERIOD 13) are calculated using the previous MOVAVE values as new data points. For example, the first predicted value (694,975.6) is the average of the data points from periods 11 and 12 (620,264 and 762328) and the moving average for period 12 (702334.7). The calculation is: $694,975 = (620,264 + 762328 + 702334.7) / 3$.

Example Using an Existing Field as a Simple Moving Average Column

The following is the same request as the example *Calculating a New Simple Moving Average Column* on page 6-22, but uses the same name for the RECAP field as the first argument in the FORECAST parameter list. The trend values do not display in the report. The actual data values for DOLLARS are followed by the predicted values in the report column.

```
DEFINE FILE GGSales
  SDATE/YYM = DATE;
  SYEAR/Y = SDATE;
  SMONTH/M = SDATE;
  PERIOD/I2 = SMONTH;
END
TABLE FILE GGSales
  SUM UNITS DOLLARS
  BY CATEGORY BY PERIOD
  WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
  ON PERIOD RECAP DOLLARS/D10.1 = FORECAST(
    DOLLARS,1,3,'MOVAVE',3);
END
```

The output is:

Category	PERIOD	Unit Sales	DOLLARS
-----	-----	-----	-----
Coffee	1	61666	801,123.0
	2	54870	682,340.0
	3	61608	765,078.0
	4	57050	691,274.0
	5	59229	720,444.0
	6	58466	742,457.0
	7	60771	747,253.0
	8	54633	655,896.0
	9	57829	730,327.0
	10	57012	724,412.0
	11	51110	620,264.0
	12	58981	762,328.0
	13	0	694,975.6
	14	0	719,879.4
	15	0	705,729.9
Food	1	54394	672,727.0
	2	54894	699,073.0
	3	52713	642,802.0
	4	58026	718,514.0
	5	53289	660,740.0
	6	58742	734,705.0
	7	60127	760,586.0
	8	55622	695,235.0
	9	55787	683,140.0
	10	57340	713,768.0
	11	57459	710,139.0
	12	57290	705,315.0
	13	0	708,398.2
	14	0	707,818.0
	15	0	708,652.3

Using an Exponential Moving Average

This method calculates an average that allows you to choose weights to apply to newer and older values.

The weight given to the newest value is k , where:

$$k = 2 / (1+n)$$

The quantity n is an integer greater than one. Increasing n increases the weight assigned to the earlier observations (or data instances) as compared to the later ones.

The next calculation of the exponential moving average (EMA) value is derived by the following formula:

$$\text{EMA} = (\text{EMA} * (1-k)) + (\text{datavalue} * k)$$

This means that the newest value from the data source is multiplied by the factor k and the current moving average is multiplied by the factor $(1-k)$. These quantities are then summed to generate the new EMA.

Note: When the data values are exhausted, the last average calculated is used as the next data value, making every predicted value a constant equal to the last calculated average.

Example

Calculating a New Exponential Moving Average Column

```
DEFINE FILE GGSALES
  SDATE/YYM = DATE;
  SYEAR/Y = SDATE;
  SMONTH/M = SDATE;
  PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
  SUM UNITS DOLLARS
  BY CATEGORY BY PERIOD
  WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
  ON PERIOD RECAP EXPAVE/D10.1= FORECAST(DOLLARS,1,3,'EXPAVE',3);
END
```

The output is:

Category	PERIOD	Unit Sales	Dollar Sales	EXPAVE
	-----	-----	-----	-----
Coffee	1	61666	801123	801,123.0
	2	54870	682340	741,731.5
	3	61608	765078	753,404.8
	4	57050	691274	722,339.4
	5	59229	720444	721,391.7
	6	58466	742457	731,924.3
	7	60771	747253	739,588.7
	8	54633	655896	697,742.3
	9	57829	730327	714,034.7
	10	57012	724412	719,223.3
	11	51110	620264	669,743.7
	12	58981	762328	716,035.8
	13	0	0	716,035.8
	14	0	0	716,035.8
	15	0	0	716,035.8
Food	1	54394	672727	672,727.0
	2	54894	699073	685,900.0
	3	52713	642802	664,351.0
	4	58026	718514	691,432.5
	5	53289	660740	676,086.3
	6	58742	734705	705,395.6
	7	60127	760586	732,990.8
	8	55622	695235	714,112.9
	9	55787	683140	698,626.5
	10	57340	713768	706,197.2
	11	57459	710139	708,168.1
	12	57290	705315	706,741.6
	13	0	0	706,741.6
	14	0	0	706,741.6
	15	0	0	706,741.6

Note:

- The number *n*, which is used to calculate the weights is 3.
- Three predicted values of EXPAVE are calculated within each value of CATEGORY. For values outside the range of the data, new PERIOD values are generated by adding the interval value (1) to the prior PERIOD value.
- There are no UNITS or DOLLARS values for the generated PERIOD values.

- Each average is computed using DOLLARS values where they exist. For predicted values beyond those points, the calculated EXPAVE values are used as new data points in the exponential average calculation.

The first EXPAVE value (801,123.0) is the same as the first DOLLARS value.

The second EXPAVE value (741,731.5) is calculated as follows. Note that because of rounding and the number of decimal places used, the value derived in this sample calculation varies slightly from the one displayed in the report output:

$n=3$ (number used to calculate weights)

$k = 2/(1+n) = 2/4 = 0.5$

$$\begin{aligned} \text{EXPAVE} &= (\text{EXPAVE} * (1-k)) + (\text{new-DOLLARS} * k) = (801123 * 0.5) + (682340 * 0.50) \\ &= 400561.5 + 341170 = 741731.5 \end{aligned}$$

The third EXPAVE value (753,404.8) is calculated as follows:

$$\begin{aligned} \text{EXPAVE} &= (\text{EXPAVE} * (1-k)) + (\text{new-DOLLARS} * k) = (741731.5 * 0.5) + (765078 * 0.50) \\ &= 370865.75 + 382539 = 753404.75 \end{aligned}$$

The predicted EXPAVE values (starting with 706,741.6) are calculated using the same formula as used to calculate the trend values. However, an exponential average is always calculated using the previous average and the new data point. Because the previous average is also used as the new data point, the predicted values are always equal to the last trend value. For example, for period 13, the previous average is 706,741.6 and this is also used as the next data point, therefore, the average is calculated as follows:

$$(706,741.6 * 0.5) + (706,741.6 * 0.5) = 706,741.6$$

$$\begin{aligned} \text{EXPAVE} &= (\text{EXPAVE} * (1-k)) + (\text{new-DOLLARS} * k) = (706741.6 * 0.5) + \\ & (706741.6 * 0.50) = 353370.8 + 353370.8 = 706741.6 \end{aligned}$$

Using a Linear Regression Equation

This method estimates values by assuming that the dependent variable (y , the new calculated values) and the independent variable (x , the sort field values) are related by the following function, which represents a straight line:

$$y = mx + b$$

The value of m represents the slope of the line, and b represents the y -intercept.

REGRESS uses a technique called Ordinary Least Squares to calculate values for m and b that minimize the sum of the squared differences between the data and the resulting line.

The following formulas show how m and b are calculated. In these formulas, n is the number of data points, the y values are the data values (dependent variable), and the x values are the sort field values (independent variable):

$$\begin{aligned} m &= (\sum xy - (\sum x * \sum y) / n) / (\sum x^2 - (\sum x)^2 / n) \\ b &= (\sum y) / n - (m * (\sum x) / n) \end{aligned}$$

Trend values as well as predicted values are calculated using the regression line equation.

Example **Calculating a New Linear Regression Field**

```
TABLE FILE CAR
PRINT MPG
BY DEALER_COST
WHERE MPG NE 0.0
   ON DEALER_COST RECAP FORMPG=FORECAST(MPG,1000,3,'REGRESS');
END
```

The output is:

DEALER_COST	MPG	FORMPG
-----	---	-----
2,886	27	25.51
4,292	25	23.65
4,631	21	23.20
4,915	21	22.82
5,063	23	22.63
5,660	21	21.83
	21	21.83
5,800	24	21.65
6,000	24	21.38
7,427	16	19.49
8,300	18	18.33
8,400	18	18.20
10,000	18	16.08
11,000	18	14.75
11,194	9	14.50
14,940	11	9.53
15,940	0	8.21
16,940	0	6.88
17,940	0	5.55

Note:

- Three predicted values of FORMPG are calculated. For values outside the range of the data, new DEALER_COST values are generated by adding the interval value (1,000) to the prior DEALER_COST value.
- There are no MPG values for the generated DEALER_COST values.

- Each FORMPG value is computed using a regression line calculated using all of the actual data values for MPG.

DEALER_COST is the independent variable (x) and MPG is the dependent variable (y). The equation is used to calculate MPGFORECAST trend and predicted values.

In this case, the equation is approximately as follows:

$$\text{FORMPG} = (-0.001323 * \text{DEALER_COST}) + 29.32$$

The predicted values are (the values are not exactly as calculated by FORECAST because of rounding, but they show the process of calculating the values):

DEALER_COST	Calculation	FORMPG
15,940	$(-0.001323 * 15,940) + 29.32$	8.23
16,940	$(-0.001323 * 16,940) + 29.32$	6.91
17,940	$(-0.001323 * 17,940) + 29.32$	5.59

FORECAST Reporting Techniques

You can use FORECAST multiple times in one request. However, the FORECAST requests must all specify the same sort field, interval, and number of predictions. Only the RECAP field, method, field used to calculate the FORECAST values, and number of points to average can change. If you change any of the other parameters, the new parameters are ignored.

If you want to move a FORECAST column in the report output, use an empty COMPUTE command for the FORECAST field as a placeholder. The data type (I, F, P, D) must be the same in the COMPUTE command and the RECAP command.

To make the report output easier to interpret, you can create a field that indicates whether the FORECAST value in each row is a predicted value. To do this, define a virtual field whose value is always a constant other than zero. Rows in the report output that represent actual records in the data source will display this constant. Rows that represent predicted values will display zero. You can also propagate this field to a HOLD file.

Example **Generating Multiple FORECAST Columns in a Request**

This example calculates moving averages and exponential averages for both the DOLLARS and BUDDOLLARS fields in the GGSales data source. The sort field, interval, and number of predictions are the same for all of the calculations.

```
DEFINE FILE GGSales
  SDATE/YYM = DATE;
  SYEAR/Y = SDATE;
  SMONTH/M = SDATE;
  PERIOD/I2 = SMONTH;
END
TABLE FILE GGSales
  SUM  DOLLARS AS 'DOLLARS' BUDDOLLARS AS 'BUDGET'
  BY  CATEGORY NOPRINT BY PERIOD AS 'PER'
  WHERE SYEAR EQ 97 AND CATEGORY EQ 'Coffee'
  ON PERIOD RECAP DOLMOVAVE/D10.1= FORECAST(DOLLARS,1,0,'MOVAVE',3);
  ON PERIOD RECAP DOLEXPAVE/D10.1= FORECAST(DOLLARS,1,0,'EXPAVE',4);
  ON PERIOD RECAP BUDMOVAVE/D10.1 = FORECAST(BUDDOLLARS,1,0,'MOVAVE',3);
  ON PERIOD RECAP BUDEXPAVE/D10.1 = FORECAST(BUDDOLLARS,1,0,'EXPAVE',4);
END
```

The output is:

PER	DOLLARS	BUDGET	DOLMOVAVE	DOLEXPAVE	BUDMOVAVE	BUDEXPAVE
---	-----	-----	-----	-----	-----	-----
1	801123	801375	801,123.0	801,123.0	801,375.0	801,375.0
2	682340	725117	741,731.5	753,609.8	763,246.0	770,871.8
3	765078	810367	749,513.7	758,197.1	778,953.0	786,669.9
4	691274	717688	712,897.3	731,427.8	751,057.3	759,077.1
5	720444	739999	725,598.7	727,034.3	756,018.0	751,445.9
6	742457	742586	718,058.3	733,203.4	733,424.3	747,901.9
7	747253	773146	736,718.0	738,823.2	751,910.3	757,999.6
8	655896	685170	715,202.0	705,652.3	733,634.0	728,867.7
9	730327	753760	711,158.7	715,522.2	737,358.7	738,824.6
10	724412	709397	703,545.0	719,078.1	716,109.0	727,053.6
11	620264	630452	691,667.7	679,552.5	697,869.7	688,413.0
12	762328	718837	702,334.7	712,662.7	686,228.7	700,582.6

Example**Moving the FORECAST Column**

The following example places the DOLLARS field after the MOVAVE field by using an empty COMPUTE command as a placeholder for the MOVAVE field. Both the COMPUTE command and the RECAP command specify formats for MOVAVE (of the same data type), but the format on the RECAP command takes precedence.

```

DEFINE FILE GGSales
  SDATE/YYM = DATE;
  SYEAR/Y = SDATE;
  SMONTH/M = SDATE;
  PERIOD/I2 = SMONTH;
END
TABLE FILE GGSales
SUM  UNITS
COMPUTE MOVAVE/D10.2 = ;
DOLLARS
  BY CATEGORY BY PERIOD
  WHERE SYEAR EQ 97 AND CATEGORY EQ 'Coffee'
  ON PERIOD RECAP MOVAVE/D10.1= FORECAST(DOLLARS,1,3,'MOVAVE',3);
END

```

The output is:

Category	PERIOD	Unit Sales	MOVAVE	Dollar Sales
-----	-----	-----	-----	-----
Coffee	1	61666	738,659.3	801123
	2	54870	734,447.5	682340
	3	61608	749,513.7	765078
	4	57050	712,897.3	691274
	5	59229	725,598.7	720444
	6	58466	718,058.3	742457
	7	60771	736,718.0	747253
	8	54633	715,202.0	655896
	9	57829	711,158.7	730327
	10	57012	703,545.0	724412
	11	51110	691,667.7	620264
	12	58981	702,334.7	762328
	13	0	694,975.6	0
	14	0	719,879.4	0
	15	0	705,729.9	0

Example

Distinguishing Data Rows From Predicted Rows

In the following example, the DATA_ROW virtual field has the value 1 for each row in the data source. It has the value zero for the predicted rows. The PREDICT field is calculated as YES for predicted rows and NO for rows containing data.

```
DEFINE FILE CAR
DATA_ROW/I1 = 1;
END
TABLE FILE CAR
  PRINT DATA_ROW
  COMPUTE PREDICT/A3 = IF DATA_ROW EQ 1 THEN 'NO' ELSE 'YES' ;
  MPG
  BY DEALER_COST
  WHERE MPG GE 20
    ON DEALER_COST RECAP FORMPG/D12.2=FORECAST(MPG,1000,3,'REGRESS');
    ON DEALER_COST RECAP MPG          =FORECAST(MPG,1000,3,'REGRESS');
  END
```

The output is:

DEALER_COST	DATA_ROW	PREDICT	MPG	FORMPG
-----	-----	-----	---	-----
2,886	1	NO	27.00	25.65
4,292	1	NO	25.00	23.91
4,631	1	NO	21.00	23.49
4,915	1	NO	21.00	23.14
5,063	1	NO	23.00	22.95
5,660	1	NO	21.00	22.21
	1	NO	21.00	22.21
5,800	1	NO	24.20	22.04
6,000	1	NO	24.20	21.79
7,000	0	YES	20.56	20.56
8,000	0	YES	19.32	19.32
9,000	0	YES	18.08	18.08

Using Functions With Temporary Fields

Any function name encountered in a DEFINE or COMPUTE expression that is not recognized as a supplied name is assumed to be a user-written function. These functions are loaded dynamically when needed. They are coded by users and reside in a library that is available at the time they are referenced.

Creating Temporary Fields Unrelated to Master Files

The temporary fields you create with the `DEFINE` and `COMPUTE` commands are tied to a specific Master File, and in the case of values calculated with the `COMPUTE` command, to a specific request. However, you can create temporary fields that are independent of either a Master File or a request using the `DEFINE FUNCTION` command.

A `DEFINE` function is a named group of calculations that use any number of input values and produce a return value.

A `DEFINE` function can be called in most of the same situations that are valid for user-written subroutines. Data types (numeric or alphanumeric) referenced in the arguments defined, and the arguments used must match. Shorter alphanumeric arguments are padded with blanks while longer alphanumeric arguments are truncated.

All calculations within the function are done in double precision. Format conversions occur only across equal signs in the assignments that define temporary fields.

Before calling a `DEFINE` function, you must issue the commands that define the function.

Syntax

How to Define a Function

```
DEFINE FUNCTION name (parameter1/format1,..., parametern/formatn)
[tempvariablea/formata = expressiona;]
.
.
.
[tempvariablex/formatx = expressionx;]
name/format = [result_expression];
END
```

where:

name

Is the name of the function. This must be the last field calculated in the function and is used to return the value of the function to the calling procedure.

format

Is the format of the value the function returns.

parameter1/format1...parametern/formatn

Are the parameter names and their formats.

If a parameter is alphanumeric, the calling argument must be alphanumeric. Shorter calling arguments are padded on the right with blanks, and longer arguments are truncated.

If a parameter is numeric, the calling argument must also be numeric. To prevent unexpected results, you must be consistent in your use of data types.

tempvariablea/formata...tempvariablex/formatx

Are temporary fields and their formats. Temporary fields hold intermediate values used in the function. You can define as many temporary fields as you need.

expressiona...expressionx

Are the expressions that calculate the temporary field values. The expressions can use parameters, constants, and other temporary fields defined in the same function.

result_expression

Is the expression that calculates the value returned by the function. The expression can use parameters, constants, and temporary fields defined in the same function.

All names defined in the body of the function are local to the function. The last field defined before the END command in the function definition must have the same name as the function and represents the return value for the function.

Reference

DEFINE Function Limits and Restrictions

- The number of functions you can define and use in a session is virtually unlimited.
- DEFINE functions are not cleared by issuing a JOIN, or any other FOCUS command, with the exception of DEFINE FUNCTION CLEAR.
- When an expression tries to use a cleared function, an error displays.
- Function names are limited to eight characters. There is no limit to the number of parameters.
- Parameter names are limited to twelve characters.
- DEFINE functions can use other DEFINE functions but cannot be used recursively.
- If you overwrite or clear a DEFINE function, a subsequent attempt to use a temporary field that refers to the function generates the following warning:
(FOC1956) DEFINE FUNCTION %1 used after CLEAR or DEFINE.
- You cannot call DEFINE functions from Dialogue Manager commands.

Example

Defining a Function

```
DEFINE FUNCTION SUBTRACT (VAL1/D8, VAL2/D8)
SUBTRACT/D8.2 = VAL1 - VAL2;
END

TABLE FILE MOVIES
PRINT TITLE LISTPR IN 35 WHOLESALEPR AND COMPUTE
PROFIT/D8.2 = SUBTRACT(LISTPR,WHOLESALEPR);
BY CATEGORY
    WHERE CATEGORY EQ 'MYSTERY' OR 'ACTION'
END
```

SUBTRACT is the name of the function. It uses local parameters VAL1 and VAL2. The output is:

CATEGORY	TITLE	LISTPR	WHOLESALEPR	PROFIT
-----	-----	-----	-----	-----
ACTION	JAWS	19.95	10.99	8.96
	ROBOCOP	19.98	11.50	8.48
	TOTAL RECALL	19.99	11.99	8.00
	TOP GUN	14.95	9.99	4.96
	RAMBO III	19.95	10.99	8.96
MYSTERY	REAR WINDOW	19.98	9.00	10.98
	VERTIGO	19.98	9.00	10.98
	FATAL ATTRACTION	29.98	15.99	13.99
	NORTH BY NORTHWEST	19.98	9.00	10.98
	DEAD RINGERS	25.99	15.99	10.00
	MORNING AFTER, THE	19.95	9.99	9.96
	PSYCHO	19.98	9.00	10.98
	BIRDS, THE	19.98	9.00	10.98
	SEA OF LOVE	59.99	30.00	29.99

Syntax

How to Query DEFINE Functions

You can display a list of all defined functions and their parameters by issuing the command:

```
? FUNCTION
```

A screen similar to the following displays:

```
FUNCTIONS CURRENTLY ACTIVE
Name      Format      Parameter      Format
-----
SUBTRACT  D8.2          VAL1           D8
              VAL2           D8
```

If you issue the ? FUNCTION command with no functions defined, the following displays:

```
NO FUNCTIONS CURRENTLY IN EFFECT
```


Syntax

How to Clear DEFINE Functions

You can clear functions by issuing the command

```
DEFINE FUNCTION {name | *} CLEAR
```

where:

name

Is the function name to clear.

*

Clears all active functions.

CHAPTER 7

Including Totals and Subtotals

Topics:

- Calculating Row and Column Totals
- Adding Section Totals and a Grand Total
- Including Subtotals
- Recalculating Values for Subtotal Rows
- Performing Calculations at Sort Field Breaks
- Suppressing Grand Totals
- Conditionally Displaying Summary Lines and Text

To help interpret detailed information in a report, you can summarize numeric information using row and column totals, grand totals, and subtotals. You can use these summary lines to clarify or highlight information in a report.

Calculating Row and Column Totals

To produce totals for rows or columns of numbers in a report, you can use the ROW-TOTAL and COLUMN-TOTAL phrases:

- ROW-TOTAL displays a new column containing the sum of all numbers in each row.
- COLUMN-TOTAL displays a final row on the report, which contains the totals for each column of numbers.

You can also use row totals and column totals in matrix reports (created by using a BY and an ACROSS in your report request), rename row and column total titles, and include calculated values in your row or column totals.

You can also create column totals using ACROSS-TOTAL. For details, see Chapter 4, *Sorting Tabular Reports*.

Syntax

How to Calculate Row and Column Totals

```
display_command fieldname AND ROW-TOTAL [alignment] [/format] [AS 'name']  
display_command fieldname AND COLUMN-TOTAL [alignment] [AS 'name']
```

where:

display_command

Is one of the following commands: PRINT, LIST, SUM, or COUNT.

fieldname

Is the name of the field for which to calculate row and/or column totals.

alignment

Specifies the alignment of the ROW-TOTAL or COLUMN-TOTAL label. Possible values are:

/R right justifies the label.

/L left justifies the label.

/C centers the label.

Note that these alignment settings are ignored in HTML output. If you are working in WebFOCUS or in the Web Interface to FOCUS, to take advantage of column alignment features, you can include the command SET STYLE=OFF in the report request or generate your output in PDF, or in another format that supports these features. For details, see Chapter 11, *Saving and Reusing Report Output*.

format

Reformats the ROW-TOTAL.

name

Is the label for the ROW-TOTAL or COLUMN-TOTAL.

You may also specify row or column totals with the ON TABLE command. Field names are optional with COLUMN-TOTAL and cannot be listed with ROW-TOTAL. Use the following syntax:

```
ON TABLE COLUMN-TOTAL [alignment] [AS 'name'] [fieldname fieldname
fieldname]
ON TABLE ROW-TOTAL [alignment] [/format] [AS 'name']
```

Reference

Usage Notes for Row and Column Totals

- If one field is summed, the format of the row total is the same as the format of the field. For instance, if the format of CURR_SAL is D12.2M, the format of the row total for CURR_SAL is also D12.2M.
- When you are summing fields with different formats, a default format of D12.2 is used for the total.

Example

Calculating Row and Column Totals

The following request illustrates the use of ROW-TOTAL and COLUMN-TOTAL:

```
TABLE FILE SALES
SUM RETURNS DAMAGED AND ROW-TOTAL AND COLUMN-TOTAL
BY PROD_CODE
END
```

The output is:

PROD_CODE	RETURNS	DAMAGED	TOTAL
-----	-----	-----	-----
B10	13	10	23
B12	4	3	7
B17	4	2	6
B20	1	2	3
C13	3	0	3
C17	0	0	0
C7	5	4	9
D12	3	2	5
E1	4	7	11
E2	9	4	13
E3	12	11	23
TOTAL	58	45	103

Example Specifying Column Totals With ON TABLE

The following request illustrates the use of COLUMN-TOTAL with the ON TABLE command:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY LAST_NAME
ON TABLE COLUMN-TOTAL
END
```

The output is:

LAST_NAME	CURR_SAL
-----	-----
BANNING	\$29,700.00
BLACKWOOD	\$21,780.00
CROSS	\$27,062.00
GREENSPAN	\$9,000.00
IRVING	\$26,862.00
JONES	\$18,480.00
MCCOY	\$18,480.00
MCKNIGHT	\$16,100.00
ROMANS	\$21,120.00
SMITH	\$13,200.00
	\$9,500.00
STEVENS	\$11,000.00
TOTAL	\$222,284.00

Example Using Row and Column Totals in a Matrix Report

The following request illustrates the use of ROW-TOTAL and COLUMN-TOTAL in a matrix report (created by using the BY and ACROSS phrases together).

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AND ROW-TOTAL AND COLUMN-TOTAL
BY BANK_NAME
ACROSS DEPARTMENT
END
```

The output is:

BANK_NAME	DEPARTMENT		TOTAL
	MIS	PRODUCTION	
-----	-----	-----	-----
	\$40,680.00	\$41,620.00	\$82,300.00
ASSOCIATED	\$21,780.00	\$42,962.00	\$64,742.00
BANK ASSOCIATION	\$27,062.00	.	\$27,062.00
BEST BANK	.	\$29,700.00	\$29,700.00
STATE	\$18,480.00	.	\$18,480.00
TOTAL	\$108,002.00	\$114,282.00	\$222,284.00

Example**Renaming Row and Column Totals in Sorted Reports (BY)**

The following request illustrates how to rename the ROW-TOTAL and COLUMN-TOTAL labels in a report that is sorted vertically:

```
TABLE FILE CAR
SUM DCOST RCOST ROW-TOTAL/C/D12 AS 'TOTAL_COST'
BY COUNTRY
ON TABLE COLUMN-TOTAL/C AS 'FINAL_TOTAL'
END
```

The output is:

COUNTRY	DEALER_COST	RETAIL_COST	TOTAL_COST
-----	-----	-----	-----
ENGLAND	37,853	45,319	83,172
FRANCE	4,631	5,610	10,241
ITALY	41,235	51,065	92,300
JAPAN	5,512	6,478	11,990
W GERMANY	54,563	64,732	119,295
FINAL_TOTAL	143,794	173,204	316,998

Example**Renaming Row and Column Totals in Sorted Reports (ACROSS)**

This request renames the ROW-TOTAL and COLUMN-TOTAL in a report that is sorted horizontally.

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
AND ROW-TOTAL AS 'TOTAL_SALARY'
ACROSS DEPARTMENT
BY CURR_JOBCODE
ON TABLE COLUMN-TOTAL AS 'FINAL_TOTAL'
END
```

The output is:

	DEPARTMENT		
	MIS	PRODUCTION	TOTAL_SALARY
CURR_JOBCODE	-----	-----	-----
A01	.	\$9,500.00	\$9,500.00
A07	\$9,000.00	\$11,000.00	\$20,000.00
A15	.	\$26,862.00	\$26,862.00
A17	\$27,062.00	\$29,700.00	\$56,762.00
B02	\$18,480.00	\$16,100.00	\$34,580.00
B03	\$18,480.00	.	\$18,480.00
B04	\$21,780.00	\$21,120.00	\$42,900.00
B14	\$13,200.00	.	\$13,200.00
FINAL_TOTAL	\$108,002.00	\$114,282.00	\$222,284.00

Example **Including Calculated Values in Row and Column Totals**

The following request illustrates the inclusion of the calculated value, PROFIT, in row and column totals.

```
TABLE FILE CAR
SUM DCOST RCOST
COMPUTE PROFIT/D12=RCOST-DCOST;
ROW-TOTAL/L/D12 AS 'TOTAL_COST'
BY COUNTRY
ON TABLE COLUMN-TOTAL/L AS 'FINAL_TOTAL'
END
```

The output is:

COUNTRY	DEALER_COST	RETAIL_COST	PROFIT	TOTAL_COST
-----	-----	-----	-----	-----
ENGLAND	37,853	45,319	7,466	90,638
FRANCE	4,631	5,610	979	11,220
ITALY	41,235	51,065	9,830	102,130
JAPAN	5,512	6,478	966	12,956
W GERMANY	54,563	64,732	10,169	129,464
FINAL_TOTAL	143,794	173,204	29,410	346,408

Adding Section Totals and a Grand Total

Frequently, reports contain detailed information that is broken down into subsections for which simple column and row totals may not provide adequate summaries. In these instances, it is more useful to look at subtotals for particular sections and a grand total at the end of the report.

You can add the following commands to your requests to create section subtotals and grand totals:

- SUB-TOTAL and SUBTOTAL
- SUMMARIZE and RECOMPUTE (used with calculated values)
- RECAP and COMPUTE

Each command produces grand totals and/or subtotals by using different numeric information. Subtotals produce totals every time a specified sort field value changes and are independent of record selection criteria. You can further control when subtotals are produced by specifying WHEN criteria (see *Conditionally Displaying Summary Lines and Text* on page 7-21). You can also suppress grand totals using the NOTOTAL command. For details, see *Suppressing Grand Totals* on page 7-19.

Example**Using Section Totals and Grand Totals**

The following request illustrates how to create a subtotal every time the department value changes and a grand total for the entire report:

```
TABLE FILE EMPLOYEE
SUM DED_AMT BY DED_CODE BY DEPARTMENT
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON DEPARTMENT SUBTOTAL
END
```

The first and last portions of the output are:

PAGE 1

DED_CODE	DEPARTMENT	BANK_ACCT	DED_AMT
CITY	MIS	40950036	\$14.01
		122850108	\$31.76
		163800144	\$82.69
*TOTAL DEPARTMENT MIS			\$128.46
	PRODUCTION	160633	\$7.42
		136500120	\$18.26
		819000702	\$60.24
*TOTAL DEPARTMENT PRODUCTION			\$85.92
FED	MIS	40950036	\$1,190.77
		122850108	\$2,699.81
		163800144	\$7,028.29
.			
.			
.			
	PRODUCTION	160633	\$103.95
		136500120	\$255.64
		819000702	\$843.31
*TOTAL DEPARTMENT PRODUCTION			\$1,202.90
TOTAL			\$41,521.46

Including Subtotals

You can use the SUBTOTAL and SUB-TOTAL commands to sum individual values, such as columns of numbers, each time a named sort field changes value.

- SUB-TOTAL displays a subtotal for all numeric values when the BY/ON field changes value and for any higher-level sort fields when their values change.
- SUBTOTAL displays a subtotal only when the specified sort field changes value. It does not give subtotals for higher-level fields.

Both SUB-TOTAL and SUBTOTAL produce grand totals. You can suppress grand totals using the NOTOTAL command. See *Suppressing Grand Totals* on page 7-19.

The subtotal is calculated every time the BY field value changes or, if WHEN criteria are applied to the BY field, every time the WHEN conditions are met.

A BY or ON phrase is required to initialize the syntax.

Syntax

How to Create Subtotals

```
{BY|ON} fieldname {SUB-TOTAL|SUBTOTAL} [MULTILINES] [AS 'text']  
[field1 [AND] field2...] [WHEN expression]
```

where:

fieldname

Must be the name of a field in a BY phrase. The number of fields to subtotal multiplied by the number of levels of subtotals counts in the number of display fields permitted for the request. For details on determining the maximum number of display fields that can be used in a request, see Chapter 1, *Creating Tabular Reports*.

SUB-TOTAL/SUBTOTAL

SUB-TOTAL displays subtotals for numeric values when the BY/ON field changes value and for any higher-level sort fields when their values change.

SUBTOTAL displays a subtotal only when the specified sort field changes value.

MULTILINES

Suppresses the printing of a subtotal line for every sort break that has only one detail line, since the subtotal value is equal to this one value. Note that MULTI-LINES is a synonym for MULTILINES.

AS '*text*'

Enables you to specify a different label. For related information, see Chapter 9, *Customizing Tabular Reports*.

field1, field2, ...

Denotes a list of specific fields to subtotal. This list overrides the default, which includes all numeric display fields.

WHEN expression

Specifies the conditional display of subtotals as determined by a Boolean expression (see *Conditionally Displaying Summary Lines and Text* on page 7-21). You must end the expression with a semicolon.

Reference

Usage Notes for Subtotals

- Use only one of the following in a request: SUB-TOTAL, SUBTOTAL, RECOMPUTE, or SUMMARIZE.
- When using a SUM or COUNT command with only one BY phrase in the request, SUB-TOTAL and SUBTOTAL produce the same result as the value of the SUM or COUNT command. However, when using a PRINT command with one BY phrase, SUBTOTAL is useful because there can be many values within a sort break.
- All subtotals are displayed up to and including the point where the sort break occurs, so that only the innermost point of subtotalling should be requested. For instance, if the BY fields are

BY AREA

BY PROD_CODE

BY DATE SUB-TOTAL

then, when AREA changes, subtotals are displayed for DATE, PROD_CODE, and AREA on three lines (one under the other).

- To suppress grand totals produced by SUB-TOTAL and SUBTOTAL, use the NOTOTAL option (see *Suppressing Grand Totals* on page 7-19).
- SUBTOTAL and SUB-TOTAL should not be used with prefix operators.

Example

Generating Subtotals

The following request illustrates how to create a subtotal for SALES every time the country value changes.

```
TABLE FILE CAR
SUM AVE.MPG AND SALES AND AVE.RETAIL_COST
BY COUNTRY SUB-TOTAL SALES
BY BODYTYPE
END
```

The output is:

COUNTRY	BODYTYPE	AVE MPG	SALES	AVE RETAIL_COST
-----	-----	----	-----	-----
ENGLAND	CONVERTIBLE	16	0	8,878
	HARDTOP	25	0	5,100
	SEDAN	10	12000	15,671
*TOTAL ENGLAND			12000	
FRANCE	SEDAN	21	0	5,610
*TOTAL FRANCE			0	
ITALY	COUPE	11	12400	19,160
	ROADSTER	21	13000	6,820
	SEDAN	21	4800	5,925
*TOTAL ITALY			30200	
JAPAN	SEDAN	14	78030	3,239
*TOTAL JAPAN			78030	
W GERMANY	SEDAN	20	88190	9,247
*TOTAL W GERMANY			88190	
FINAL_TOTAL			208420	

Example

Comparing SUB-TOTAL and SUBTOTAL

The following request illustrates how to create a subtotal for the numeric fields DED_AMT and GROSS when the department value changes, and for the higher-level sort field (DED_CODE) when its value changes:

```
TABLE FILE EMPLOYEE
SUM DED_AMT GROSS BY DED_CODE BY DEPARTMENT
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON DEPARTMENT SUB-TOTAL
END
```

If you use SUBTOTAL instead of SUB-TOTAL, totals for DED_AMT and GROSS display only when the DEPARTMENT value changes.

The first and last portions of output are:

PAGE 1

DED_CODE	DEPARTMENT	BANK_ACCT	DED_AMT	GROSS
CITY	MIS	40950036	\$14.01	\$6,099.50
		122850108	\$31.76	\$9,075.00
		163800144	\$82.69	\$22,013.77
*TOTAL DEPARTMENT MIS			\$128.46	\$37,188.27
	PRODUCTION	160633	\$7.42	\$2,475.00
		136500120	\$18.26	\$9,129.99
		819000702	\$60.24	\$17,094.00
*TOTAL DEPARTMENT PRODUCTION			\$85.92	\$28,698.99
*TOTAL DED_CODE CITY			\$214.38	\$65,887.26
.				
.				
.				
*TOTAL DEPARTMENT MIS			\$1,798.40	\$37,188.27
	PRODUCTION	160633	\$103.95	\$2,475.00
		136500120	\$255.64	\$9,129.99
		819000702	\$843.31	\$17,094.00
*TOTAL DEPARTMENT PRODUCTION			\$1,202.90	\$28,698.99
*TOTAL DED_CODE STAT			\$3,001.30	\$65,887.26
TOTAL			\$41,521.46	\$461,210.81

Recalculating Values for Subtotal Rows

You can use the SUMMARIZE and RECOMPUTE commands instead of SUB-TOTAL and SUBTOTAL to recalculate the result of a COMPUTE command.

SUMMARIZE is similar to SUB-TOTAL in that it recomputes values at every sort break. RECOMPUTE is similar to SUBTOTAL in that it recalculates only at the specified sort break.

SUMMARIZE recomputes grand totals for the entire report. If you wish to suppress the grand totals you can include the NOTOTAL command in your request. See *Suppressing Grand Totals* on page 7-19.

A BY or ON phrase is required to initialize the syntax.

Syntax

How to Subtotal Calculated Values

```
{BY|ON} fieldname {SUMMARIZE|RECOMPUTE} [MULTILINES] [AS 'text']  
[field1 [AND] field2...] [WHEN expression;
```

where:

fieldname

Must be the name of a field in a BY phrase. The number of fields to subtotal multiplied by the number of levels of subtotals counts in the number of display fields permitted for the request. For details on determining the maximum number of display fields that can be used in a request, see Chapter 1, *Creating Tabular Reports*.

SUMMARIZE|RECOMPUTE

SUMMARIZE recomputes values at every sort break.

RECOMPUTE recalculates values only at the specified sort break.

MULTILINES

Suppresses the printing of a subtotal line for every sort break that has only one detail line, since the subtotal value is equal to this one value. Note that MULTI-LINES is a synonym for MULTILINES.

You can also suppress grand totals using the NOTOTAL command, as described in *Suppressing Grand Totals* on page 7-19.

AS 'text'

Enables you to specify a different label. For related information, see Chapter 9, *Customizing Tabular Reports*.

field1, field2, ...

Denotes a list of specific fields to be subtotaled after the RECOMPUTE or SUMMARIZE. This list overrides the default, which includes all numeric display fields.

WHEN expression

Specifies the conditional display of subtotals based on a Boolean expression (see *Conditionally Displaying Summary Lines and Text* on page 7-21). You must end the expression with a semicolon.

You may also generate subtotals for the recalculated values with the ON TABLE command. Use the following syntax:

ON TABLE SUMMARIZE

Reference

Usage Notes for SUMMARIZE and RECOMPUTE

- Like SUBTOTAL and SUB-TOTAL, SUMMARIZE and RECOMPUTE should not be used with prefix operators.
- You can specify WHEN criteria with SUMMARIZE or RECOMPUTE to conditionally display subtotals for calculated values.
- Use only one of the following in a request: SUB-TOTAL, SUBTOTAL, SUMMARIZE, or RECOMPUTE.

Example Using SUMMARIZE

The following request illustrates the use of SUMMARIZE to recalculate DG_RATIO at the specified sort break, DEPARTMENT, and for the higher-level sort break, PAY_DATE:

```
TABLE FILE EMPLOYEE
SUM GROSS DED_AMT AND COMPUTE
DG_RATIO/F4.2=DED_AMT/GROSS;
BY HIGHEST PAY_DATE BY DEPARTMENT
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON DEPARTMENT SUMMARIZE
END
```

The first and last portions of the output are:

PAGE1

PAY_DATE	DEPARTMENT	BANK_ACCT	GROSS	DED_AMT	DG_RATIO
82/08/31	MIS	40950036	\$1,540.00	\$725.35	.47
		122850108	\$1,815.00	\$1,261.42	.69
		163800144	\$2,255.00	\$1,668.70	.74
*TOTAL DEPARTMENT MIS			\$5,610.00	\$3,655.47	.65
	PRODUCTION	160633	\$2,475.00	\$1,427.25	.58
		136500120	\$1,342.00	\$522.28	.39
		819000702	\$2,238.50	\$1,746.04	.78
*TOTAL DEPARTMENT PRODUCTION			\$6,055.50	\$3,695.57	.61
*TOTAL PAY_DATE 82/08/31			\$11,665.50	\$7,351.04	.63
.					
.					
.					
81/12/31	MIS	163800144	\$2,147.75	\$1,406.78	.66
*TOTAL DEPARTMENT MIS			\$2,147.75	\$1,406.78	.66
*TOTAL PAY_DATE 81/12/31			\$2,147.75	\$1,406.78	.66
81/11/30	MIS	163800144	\$2,147.75	\$1,406.78	.66
*TOTAL DEPARTMENT MIS			\$2,147.75	\$1,406.78	.66
*TOTAL PAY_DATE 81/11/30			\$2,147.75	\$1,406.78	.66
TOTAL			\$65,887.26	\$41,521.46	.63

Tip:
If you used SUB-TOTAL or SUBTOTAL rather than SUMMARIZE, the values of DG_RATIO would be added.

Example**Using RECOMPUTE**

The following request illustrates the use of RECOMPUTE to recalculate DG_RATIO only at the specified sort break, DEPARTMENT.

```
TABLE FILE EMPLOYEE
SUM GROSS DED_AMT AND COMPUTE
DG_RATIO/F4.2=DED_AMT/GROSS;
BY HIGHEST PAY_DATE BY DEPARTMENT
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON DEPARTMENT RECOMPUTE
END
```

The output is:

PAGE 1

PAY_DATE	DEPARTMENT	BANK_ACCT	GROSS	DED_AMT	DG_RATIO
82/08/31	MIS	40950036	\$1,540.00	\$725.35	.47
		122850108	\$1,815.00	\$1,261.42	.69
		163800144	\$2,255.00	\$1,668.70	.74
*TOTAL DEPARTMENT MIS			\$5,610.00	\$3,655.47	.65
	PRODUCTION	160633	\$2,475.00	\$1,427.25	.58
		136500120	\$1,342.00	\$522.28	.39
		819000702	\$2,238.50	\$1,746.04	.78
*TOTAL DEPARTMENT PRODUCTION			\$6,055.50	\$3,695.57	.61
.					
.					
.					
81/12/31	MIS	163800144	\$2,147.75	\$1,406.78	.66
*TOTAL DEPARTMENT MIS			\$2,147.75	\$1,406.78	.66
81/11/30	MIS	163800144	\$2,147.75	\$1,406.78	.66
*TOTAL DEPARTMENT MIS			\$2,147.75	\$1,406.78	.66
TOTAL			\$65,887.26	\$41,521.46	.63

Performing Calculations at Sort Field Breaks

You can use the RECAP and COMPUTE commands to create subtotal values in a calculation. The subtotal values are not displayed; only the result of the calculation is shown on the report.

A BY or ON phrase is required to initialize the syntax.

Syntax

How to Use Subtotals in Calculations

Both the RECAP and COMPUTE commands have similar syntax to other total and subtotal commands.

```
{BY|ON} fieldname1 {RECAP|COMPUTE} fieldname2[/format] = expression;  
[WHEN expression;]
```

where:

fieldname1

Is the field in the BY phrase. Each time the BY field changes value, a new recap value is calculated.

fieldname2

Is the field name that contains the result of the expression.

/format

Can be any valid format. The default is D12.2.

expression

Can be any valid expression, described in Chapter 8, *Using Expressions*. You must end the expression with a semicolon.

WHEN expression

Is for use with RECAP only. Specifies the conditional display of RECAP lines as determined by a Boolean expression (see *Conditionally Displaying Summary Lines and Text* on page 7-21). You must end the expression with a semicolon.

Reference

Usage Notes for RECAP and COMPUTE

- RECAP uses the current value of the named sort field, the current subtotal values of any computational fields that appear as display fields, or the last value for alphanumeric fields.
- The field names in the expression must be fields that appear on the report. That is, they must be display fields or sort control fields.
- Each RECAP value displays on a separate line. However, if the request contains a RECAP command and SUBFOOT text, the RECAP value displays only in the SUBFOOT text and must be specified in the text using a spot marker. (For details, see Chapter 9, *Customizing Tabular Reports*.)
- The calculations in a RECAP or COMPUTE can appear anywhere under the control break along with any text. (For details, see Chapter 9, *Customizing Tabular Reports*.)
- The word RECAP may not be specified more than seven times. However, more than seven RECAP calculations are permitted. Use the following syntax:

```
ON fieldname RECAP field1/format= ... ;  
field2/format= ... ;  
.  
.  
.
```

Example Using RECAP

The following request illustrates the use of RECAP (DEPT_NET) to determine net earnings for each department:

```
TABLE FILE EMPLOYEE
SUM DED_AMT AND GROSS
BY DEPARTMENT BY PAY_DATE
ON DEPARTMENT RECAP DEPT_NET/D8.2M = GROSS-DED_AMT;
WHEN PAY_DATE GT 820101
END
```

The output is:

PAGE 1

DEPARTMENT	PAY_DATE	DED_AMT	GROSS
	-----	-----	-----
MIS	81/11/30	\$1,406.78	\$2,147.75
	81/12/31	\$1,406.78	\$2,147.75
	82/01/29	\$1,740.88	\$3,247.75
	82/02/26	\$1,740.88	\$3,247.75
	82/03/31	\$1,740.88	\$3,247.75
	82/04/30	\$3,386.76	\$5,890.84
	82/05/28	\$3,954.39	\$6,649.51
	82/06/30	\$4,117.07	\$7,460.00
	82/07/30	\$4,117.07	\$7,460.00
	82/08/31	\$4,575.76	\$9,000.00
** DEPT_NET		\$22,311.88	
PRODUCTION	81/11/30	\$141.66	\$833.33
	81/12/31	\$141.66	\$833.33
	82/01/29	\$1,560.10	\$3,705.84
	82/02/26	\$2,061.70	\$4,959.84
	82/03/31	\$2,061.70	\$4,959.84
	82/04/30	\$2,061.70	\$4,959.84
	82/05/28	\$3,483.89	\$7,048.84
	82/06/30	\$3,483.89	\$7,048.84
	82/07/30	\$3,483.89	\$7,048.84
	82/08/31	\$4,911.14	\$9,523.84
** DEPT_NET		\$27,531.03	

Example

Using Multiple RECAP Commands

You can include multiple RECAP or COMPUTE commands in a request. This option enables you to perform different calculations at different control breaks.

The following request illustrates the use of multiple RECAP commands:

```
TABLE FILE SALES
SUM UNIT_SOLD AND RETURNS
WHERE AREA EQ 'U'
BY DATE BY AREA BY PROD_CODE
ON DATE RECAP
DATE_RATIO=RETURNS/UNIT_SOLD;
ON AREA UNDER-LINE RECAP
AREA_RATIO=RETURNS/UNIT_SOLD;
END
```

The first page of output is:

DATE	AREA	PROD_CODE	UNIT_SOLD	RETURNS

10/17	U	B10	30	2
		B17	20	2
		B20	15	0
		C17	12	0
		D12	20	3
		E1	30	4
		E3	35	4
** AREA_RATIO				.09
** DATE_RATIO				.09

10/18	U	B10	13	1
** AREA_RATIO				.08
** DATE_RATIO				.08

10/19	U	B12	29	1
** AREA_RATIO				.03
** DATE_RATIO				.03

Suppressing Grand Totals

You can use the NOTOTAL command to suppress grand totals on a report.

Suppressing the grand total is useful when there is only one value at a sort break, since the grand total value is equal to that one value. Using the NOTOTAL command prevents the report from displaying a grand total line for every sort break that has only one detail line. You can also suppress subtotals using the MULTILINES command. For details, see *How to Create Subtotals* on page 7-8.

Syntax

How to Suppress Grand Totals

To suppress grand totals, add the following syntax to your request:

```
ON TABLE NOTOTAL
```

Example

Suppressing Grand Totals

The following request includes the NOTOTAL phrase to suppress grand totals for CURR_SAL, GROSS, and DED_AMT.

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AND GROSS AND DED_AMT
BY EMP_ID
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON BANK_ACCT SUB-TOTAL
ON TABLE NOTOTAL
END
```

The output is:

PAGE 1

EMP_ID	BANK_ACCT	CURR_SAL	GROSS	DED_AMT
117593129	40950036	\$18,480.00	\$6,099.50	\$2,866.20
*TOTAL	40950036	\$18,480.00	\$6,099.50	\$2,866.20
*TOTAL	117593129	\$18,480.00	\$6,099.50	\$2,866.20
119329144	160633	\$29,700.00	\$2,475.00	\$1,427.25
*TOTAL	160633	\$29,700.00	\$2,475.00	\$1,427.25
*TOTAL	119329144	\$29,700.00	\$2,475.00	\$1,427.25
123764317	819000702	\$26,862.00	\$17,094.00	\$11,949.53
*TOTAL	819000702	\$26,862.00	\$17,094.00	\$11,949.53
*TOTAL	123764317	\$26,862.00	\$17,094.00	\$11,949.53
326179357	122850108	\$21,780.00	\$9,075.00	\$6,307.12
*TOTAL	122850108	\$21,780.00	\$9,075.00	\$6,307.12
*TOTAL	326179357	\$21,780.00	\$9,075.00	\$6,307.12
451123478	136500120	\$16,100.00	\$9,129.99	\$3,593.93
*TOTAL	136500120	\$16,100.00	\$9,129.99	\$3,593.93
*TOTAL	451123478	\$16,100.00	\$9,129.99	\$3,593.93
818692173	163800144	\$27,062.00	\$22,013.77	\$15,377.41
*TOTAL	163800144	\$27,062.00	\$22,013.77	\$15,377.41
*TOTAL	818692173	\$27,062.00	\$22,013.77	\$15,377.41

Conditionally Displaying Summary Lines and Text

In addition to using summary lines to control the look and content of your report, you can specify WHEN criteria to control the conditions under which summary lines appear for each vertical (BY) sort field value. WHEN is supported with SUBTOTAL, SUB-TOTAL, SUMMARIZE, RECOMPUTE, and RECAP. For details on the WHEN phrase, see Chapter 9, *Customizing Tabular Reports*.

Example

Conditionally Displaying Summary Lines and Text

In a sales report that covers four regions (Midwest, Northeast, Southeast, and West), you may want to only display a subtotal when total dollar sales are greater than 11500000. The following request accomplishes this by including criteria to trigger the display of a subtotal and subfooting text only when dollar sales exceed 11500000.

```
TABLE FILE GGSALES
SUM UNITS DOLLARS
BY REGION
BY CATEGORY
ON REGION SUBTOTAL
WHEN DOLLARS GT 11500000
SUBFOOT
"The total for the <REGION region is less than 11500000."
WHEN DOLLARS LT 11500000
END
```

The output is:

Region	Category	Unit Sales	Dollar Sales
-----	-----	-----	-----
Midwest	Coffee	332777	4178513
	Food	341414	4338271
	Gifts	230854	2883881
The total for the Midwest region is less than 11500000.			
Northeast	Coffee	335778	4164017
	Food	353368	4380004
	Gifts	227529	2848289
The total for the Northeast region is less than 11500000.			
Southeast	Coffee	350948	4415408
	Food	349829	4308731
	Gifts	234455	2986240
*TOTAL Southeast		935232	11710379
West	Coffee	356763	4473527
	Food	340234	4202338
	Gifts	235042	2977092
*TOTAL West		932039	11652957
FINAL_TOTAL		3688991	46156311

CHAPTER 8

Using Expressions

Topics:

- Using Expressions in Commands and Phrases
- Types of Expressions
- Creating a Numeric Expression
- Creating a Date or Date-Time Expression
- Creating a Character Expression
- Creating a Logical Expression
- Creating a Conditional Expression

An expression combines field names, constants, and operators in a calculation that returns a single value. You can use an expression in a variety of commands to assign a value to a temporary field or Dialogue Manager amper variable, or use it in screening. You can build increasingly complex expressions by combining simpler ones.

When you write an expression, you can specify the operation yourself, or you can use one of the many supplied functions that perform specific calculations or data manipulation. These functions operate on one or more arguments and return a single value as a result. To use a function, you simply call it. For details about functions, see the *Using Functions* manual.

Using Expressions in Commands and Phrases

You can use an expression in various commands and phrases. An expression may not exceed 40 lines and must end with a semicolon, except in WHERE and WHEN phrases, in which the semicolon is optional.

The commands that support expressions, and their basic syntax, are summarized here. For complete syntax with an explanation see the applicable documentation.

You can use an expression when you:

- Create a temporary field, and assign a value to that field. The field can be created in a Master File using the DEFINE attribute, or created using a DEFINE or COMPUTE command:

- DEFINE command preceding a report request:

```
DEFINE FILE filename
    filename[/format] = expression;
    .
    .
    .
END
```

- DEFINE attribute in a Master File:

```
DEFINE filename[/format] = expression;$
```

- COMPUTE command in a report request:

```
COMPUTE filename[/format] = expression;
```

- Define record selection criteria and criteria that control report formatting.

```
{WHERE|IF} logical_expression[;]
WHEN logical_expression[;]
```

- Determine branching in Dialogue Manager or assign a value to a Dialogue Manager amper variable.

```
-IF logical_expression [THEN] GOTO label1 [ELSE GOTO label2];
-SET &name = expression;
```

- Perform a calculation with the RECAP command in the Financial Modeling Language (FML).

```
RECAP name [(n)] [/format] = expression;
```


Types of Expressions

An expression can be one of the following:

- **Numeric.** Use numeric expressions to perform calculations that use numeric constants (integer or decimal) and fields. For example, you can write an expression to compute the bonus for each employee by multiplying the current salary by the desired percentage as follows:

```
COMPUTE BONUS/D12.2 = CURR_SAL * 0.05 ;
```

A numeric expression returns a numeric value. For details, see *Creating a Numeric Expression* on page 8-4.

- **Date.** Use date expressions to perform numeric calculations on dates. For example, you can write an expression to determine when a customer can expect to receive an order by adding the number of days in transit to the date on which you shipped the order as follows:

```
COMPUTE DELIVERY/MDY = SHIPDATE + 5 ;
```

There are two types of date expressions:

- Date expressions, which return a date, a component of a date, or an integer that represents the number of days, months, quarters, or years between two dates. For details, see *Creating a Date or Date-Time Expression* on page 8-7.
- Date-time expressions, which you can create using a variety of specialized date-time functions, each of which returns a different kind of value. For details about these functions, see the *Developing Applications* manual.
- **Character.** Use character expressions to manipulate alphanumeric constants or fields. For example, you can write an expression to extract the first initial from an alphanumeric field as follows:

```
COMPUTE FIRST_INIT/A1 = EDIT (FIRST_NAME, '9$$$$$$$$$') ;
```

A character expression returns an alphanumeric value. For details, see *Creating a Character Expression* on page 8-12.

- **Logical.** Use logical expressions to evaluate the relationship between two values. A logical expression returns TRUE or FALSE. For details, see *Creating a Logical Expression* on page 8-14.
- **Conditional.** Use conditional expressions to assign values based on the result of logical expressions. A conditional expression (IF ... THEN ... ELSE) returns a numeric or alphanumeric value. For details, see *Creating a Conditional Expression* on page 8-16.

Expressions and Field Formats

When you use an expression to assign a value to a field, make sure that you give the field a format that is consistent with the value returned by the expression. For example, if you use a character expression to concatenate a first name and last name and assign it to the field FULL_NAME, make sure you define the field as character (that is, alphanumeric or text).

Example

Assigning a Field Format of Sufficient Length

The following example contains a character expression that concatenates the first name and last name to derive the full name. It assigns the field FULL_NAME an alphanumeric format of sufficient length to accommodate the concatenated name:

```
DEFINE FILE EMPLOYEE
FULL_NAME/A25 = FIRST_NAME | LAST_NAME;
END
TABLE FILE EMPLOYEE
PRINT FULL_NAME
WHERE LAST_NAME IS 'BLACKWOOD'
END
```

The output is:

```
FULL_NAME
-----
ROSEMARIE BLACKWOOD
```

Creating a Numeric Expression

A numeric expression performs a calculation that uses numeric constants, fields, operators, or functions to return a numeric value. When you use a numeric expression to assign a value to a field, that field must have a numeric format. The default format is D12.2.

A numeric expression can consist of the following components, shown below in **bold**:

- A numeric constant. For example:
`COMPUTE COUNT/I2 = 1 ;`
- A numeric field. For example:
`COMPUTE RECOUNT/I2 = COUNT ;`
- Two numeric constants or fields joined by an arithmetic operator. For example:
`COMPUTE BONUS/D12.2 = CURR_SAL * 0.05 ;`
For a list of arithmetic operators, see *Arithmetic Operators* on page 8-5.
- A numeric function. For example:
`COMPUTE LONGEST_SIDE/D12.2 = MAX (WIDTH, HEIGHT) ;`
- Two or more numeric expressions joined by an arithmetic operator. For example:
`COMPUTE PROFIT/D12.2 = (RETAIL_PRICE - UNIT_COST) * UNIT_SOLD ;`
Note the use of parentheses to change the order of evaluation of the expression. For information on the order in which numeric operations are performed, see *Order of Evaluation* on page 8-5.

Before they are used in calculations, all numeric values are converted to double-precision floating-point format. The result is then converted to the specified field format. In some cases the conversion may result in a rounding difference.

If a number is too large (greater than 10^{75}) or too small (less than 10^{-75}), you receive an Overflow or Underflow warning, and asterisks display for the field value.

For detailed information on rounding behavior for numeric data formats, see the *Describing Data* manual.

Reference

Arithmetic Operators

The following list shows the arithmetic operators you can use in an expression:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**

Note: If you attempt to divide by 0, the value of the expression is 0. Multiplication and exponentiation are not supported for date expressions of any type. To isolate part of a date, use a simple assignment command.

For related information, see *Order of Evaluation* on page 8-5.

Order of Evaluation

Numeric expressions are evaluated in the following order:

1. Exponentiation.
2. Division and multiplication.
3. Addition and subtraction.

When operators are at the same level, they are evaluated from left to right. Because expressions in parentheses are evaluated before any other expression, you can use parentheses to change this predefined order. For example, the following expressions yield different results because of parentheses:

```

COMPUTE PROFIT/D12.2 = RETAIL_PRICE - UNIT_COST * UNIT_SOLD ;
COMPUTE PROFIT/D12.2 = (RETAIL_PRICE - UNIT_COST) * UNIT_SOLD ;

```

In the first expression, UNIT_SOLD is first multiplied by UNIT_COST, and the result is subtracted from RETAIL_PRICE. In the second expression, UNIT_COST is first subtracted from RETAIL_PRICE, and that result is multiplied by UNIT_SOLD.

Note: Two operators cannot appear consecutively. The following expression is invalid:

```
a * -1
```

To make it valid, you must add parentheses:

```
a* (-1)
```

Example

Controlling the Order of Evaluation

The order of evaluation can affect the result of an expression. Suppose you want to determine the dollar loss in retail sales attributed to the return of damaged items. You could issue the following request:

```
TABLE FILE SALES
PRINT RETAIL_PRICE RETURNS DAMAGED
COMPUTE RETAIL_LOSS/D12.2 = RETAIL_PRICE * RETURNS + DAMAGED;
BY PROD_CODE
WHERE PROD_CODE IS 'E1';
END
```

The calculation

```
COMPUTE RETAIL_LOSS/D12.2 = RETAIL_PRICE * RETURNS + DAMAGED;
```

gives an incorrect result because RETAIL_PRICE is first multiplied by RETURNS, and then the result is added to DAMAGED. The correct result is achieved by adding RETURNS to DAMAGED, then multiplying the result by RETAIL_PRICE.

You can change the order of evaluation by enclosing expressions in parentheses. An expression in parentheses is evaluated before any other expression. You may also use parentheses to improve readability.

Using parentheses, the correct syntax for the preceding calculation is:

```
COMPUTE RETAIL_LOSS/D12.2 = RETAIL_PRICE * (RETURNS + DAMAGED);
```

The output is:

PROD_CODE	RETAIL_PRICE	RETURNS	DAMAGED	RETAIL_LOSS
-----	-----	-----	-----	-----
E1	\$.89	4	7	10.56

Creating a Date or Date-Time Expression

A date or date-time expression performs a numeric calculation that involves dates.

A *date* expression returns a date, a date component, or an integer that represents the number of days, months, quarters, or years between two dates. You can write a date expression directly that consists of:

- A date constant. For example:

```
COMPUTE END_DATE/MDYY = 'FEB 29 2000';
```

Single quotation marks around the date constant are required.

- A date field. For example:

```
COMPUTE NEWDATE/YMD = START_DATE;
```

- An alphanumeric, integer, or packed decimal format field, with date edit options. For example, in the second COMPUTE command, OLDDATE is a date expression:

```
COMPUTE OLDDATE/I6YMD = 980307;
```

```
COMPUTE NEWDATE/YMD DFC 19 YRT 10 = OLDDATE;
```

- A calculation that uses an arithmetic operator or date function to return a date. Use a numeric operator only with date formats (formerly called Smart dates). The following example first converts the integer date HIRE_DATE (format I6YMD) to the date format CONVERTED_HDT (format YMD). It then adds 30 days to CONVERTED_HDT:

```
COMPUTE CONVERTED_HDT/YMD = HIRE_DATE;
```

```
HIRE_DATE_PLUS_THIRTY/YMD = CONVERTED_HDT + 30;
```

- A calculation that uses a numeric operator or date function to return an integer that represents the number of days, months, quarters, or years between two dates. The following example uses the date function YMD to calculate the difference (number of days) between an employee's hire date and the date of his first salary increase:

```
COMPUTE DIFF/I4 = YMD (HIRE_DATE,FST.DAT_INC);
```

A *date-time* expression returns date and time components. You can create these expressions using a variety of supplied date-time functions. For details, see the *Using Functions* manual.

Formats for Date Values

You can work with dates in one of two ways:

- **In date format.** The value is treated as an integer that represents the number of days between the date value and a base date. There are two base dates for date formats: YMD and YYMD formats have a base date of December 31, 1900; and YM and YYM formats have a base date of January, 1901. When displayed, the integer value is converted to the corresponding date in the format specified for the field. The format can be specified in either the Master File or in the command that uses an expression to assign a value to the field. These were previously referred to as smart date formatted fields.
- **In integer, packed decimal, or alphanumeric format with date edit options.** The value is treated as an integer, a packed decimal, or an alphanumeric string. When displayed, the value is formatted as a date. These were previously referred to as old date formatted fields.

You can convert a date in one format to a date in another format simply by assigning one to the other. For example, the following assignments take a date stored as an alphanumeric field, formatted with date edit options, and convert it to a date stored as a temporary date field:

```
COMPUTE ALPHADATE/A6MDY = '120599' ;  
REALDATE/MDY = ALPHADATE ;
```

Reference

Base Dates for Date Formats

The following table shows the base date for each supported date format:

Format	Base Date
YMD and YYMD	1900/12/31
YM and YYM	1901/01
YQ and YYQ	1901 Q1
JUL and YYJUL	1900/365

Note that the base date used for the functions DA and DT is December 31, 1899. For details on date functions, see the *Using Functions* manual.

Reference

Impact of Date Formats on Storage and Display

The following table illustrates how the field format affects storage and display:

	Date Format (For example: MDYY)		Integer, Packed, Decimal, or Alphanumeric Format (For example: A8MDYY)	
Value	Stored	Displayed	Stored	Displayed
February 28, 1999	35853	02/28/1999	02281999	02/28/1999
March 1, 1999	35854	03/01/1999	03011999	03/01/1999

Performing Calculations on Dates

The format of a field determines how you can use it in a date expression. Calculations on dates in date format can incorporate numeric operators as well as numeric functions. Calculations on dates in integer, packed, decimal, or alphanumeric format require the use of date functions; numeric operators return an error message or incorrect result.

A full set of functions is supplied with your software, enabling you to manipulate dates in integer, packed decimal, and alphanumeric format. For details on date functions, see the *Using Functions* manual.

Example

Calculating Dates

Assume that your company maintains a SHIPPING database. The following example calculates how many days it takes the shipping department to fill an order by subtracting the date on which an item is ordered, the ORDER_DATE, from the date on which it is shipped, SHIPDATE:

```
COMPUTE TURNAROUND/I4 = SHIP_DATE - ORDER_DATE;
```

An item ordered on February 28, 1999, and shipped on March 1, 1999, results in a difference of one day. However, if the SHIP_DATE and ORDER_DATE fields have an integer format, the result of the calculation (730000) is incorrect, since you cannot use the numeric operator minus (-) with that format.

The following table shows how the field format affects the result:

	Value in Date Format	Value in Integer Format
SHIP_DATE = March 1, 1999	35854	03011999
ORDER_DATE = February 28, 1999	35853	02281999
TURNAROUND	1	730000

To obtain the correct result using fields in integer, packed, decimal, or alphanumeric format, use the date function MDY, which returns the difference between two dates in the form month-day-year. Using the function MDY, you can calculate TURNAROUND as follows:

```
COMPUTE TURNAROUND/I4 = MDY(ORDER_DATE, SHIP_DATE);
```

Cross-Century Dates With DEFINE and COMPUTE

You can use an expression in a DEFINE or COMPUTE command, or in a DEFINE attribute in a Master File, that implements the sliding window technique for cross-century date processing. The parameters DEFCENT and YRTHRESH provide a means of interpreting the century if the first two digits of the year are not provided elsewhere. If the first two digits are provided, they are simply accepted.

For details on the sliding window technique, and the use of expressions in this technique, see the topic on cross-century dates in the *Developing Applications* manual.

Returned Field Format Selection

A date expression always returns a number. That number may represent a date or the number of days, months, quarters, or years between two dates. When you use a date expression to assign a value to a field, the format selected for the field determines how the result is returned.

Example

Selecting the Format of a Returned Field

Consider the following commands, assuming that SHIP_DATE and ORDER_DATE are date-formatted fields. The first command calculates how many days it takes a shipping department to fill an order by subtracting the date on which an item is ordered, ORDER_DATE, from the date on which it is shipped, SHIP_DATE. The second command calculates a delivery date by adding five days to the date on which the order is shipped.

```
COMPUTE TURNAROUND/I4 = SHIP_DATE - ORDER_DATE;  
COMPUTE DELIVERY/MDY = SHIP_DATE + 5;
```

In the first command, the date expression returns the *number* of days it takes to fill an order; therefore, the associated field, TURNAROUND, must have an integer format. In the second command, the date expression returns the *date* on which the item will be delivered; therefore, the associated field, DELIVERY, must have a date format.

Using a Date Constant in an Expression

When you use a date constant in a calculation with a field in date format, you must enclose it in single quotation marks; otherwise, it is interpreted as the number of days between the constant and the base date (December 31, 1900, or January 1, 1901). For example, if 022829 were not enclosed in quotation marks, the value would be interpreted as the 22,899th day after 12/31/1900, rather than as February 28, 1999.

Example

Initializing a Field With a Date Constant

The following command initializes START_DATE with the date constant 02/28/99:

```
COMPUTE START_DATE/MDY = '022899';
```

The following command calculates the number of days elapsed since January 1, 1999:

```
COMPUTE YEAR_TO_DATE/I4 = CURR_DATE - 'JAN 1 1999' ;
```


Extracting a Date Component

Date components include days, months, quarters, or years. You can write an expression that extracts a component from a field in date format. However, you cannot write an expression that extracts days, months, or quarters from a date that does not have these components. For example, you cannot extract a month from a date in YY format, which represents only the number of years.

Example

Extracting the Month Component From a Date

The following example extracts the month component from SHIP_DATE, which has the format MDYY:

```
COMPUTE SHIP_MONTH/M = SHIP_DATE;
```

If SHIP_DATE has the value March 1, 1999, the above expression returns the value 03 for SHIP_MONTH.

A calculation on a date component automatically produces a valid value for the desired component. For example, if the current value of SHIP_MONTH is 03, the following expression correctly returns the value 06:

```
COMPUTE ADD_THREE/M = SHIPMONTH + 3;
```

If the addition of months results in an answer greater than 12, the months are adjusted correctly (for example, 11 + 3 is 2, not 14).

Combining Fields With Different Formats in an Expression

When using fields in date format, you can combine fields with a different order of components within the same expression. In addition, you can assign the result of a date expression to a field with a different order of components from the fields in the expression.

You cannot, however, write an expression that combines dates in date format with dates in integer, packed, decimal or character format.

Example

Combining Fields With Format YYMD and MDY

Consider the two fields DATE_PAID and DUE_DATE. DATE_PAID has the format YYMD and DUE_DATE has the format MDY. You can combine these two fields in an expression to calculate the number of days that a payment is late:

```
COMPUTE DAYS_LATE/I4 = DATE_PAID - DUE_DATE;
```

Example

Assigning a Different Order of Components to a Returned Field

Consider the field DATE_SOLD. This field contains the date on which an item is sold, in YYMD format. The following expression adds seven days to DATE_SOLD to determine the last date on which the item can be returned. It then assigns the result to a field with DMY format:

```
COMPUTE RETURN_BY/DMY = DATE_SOLD + 7;
```

Creating a Character Expression

A character expression uses alphanumeric constants, fields, concatenation operators, or functions to derive an alphanumeric value. When you use a character expression to assign a value to a field, you must give that field an alphanumeric format.

A character expression can consist of:

- An alphanumeric constant (character string) enclosed in single quotation marks. For example:

```
COMPUTE STATE/A2 = 'NY';
```

- A combination of alphanumeric fields and/or constants joined by the concatenation operator. For example:

```
DEFINE FILE EMPLOYEE TITLE/A19 = 'DR. ' | LAST_NAME;  
END
```

- An alphanumeric function. For example:

```
DEFINE FILE EMPLOYEE INITIAL/A1 = EDIT(FIRST_NAME, '9$$$$$$$$$');  
END
```

Embedding a Quotation Mark in a Quote-Delimited Literal String

Under certain conditions, you can use quote-delimited strings containing embedded quotation marks. Two contiguous quotes within a quote-delimited string are treated as a single literal quote.

You can use quote-delimited strings in the following:

- WHERE and IF criteria containing multiple quotes.
- WHERE criteria containing: *fieldname* {IS, IS-NOT, IN, IN FILE, or NOT IN FILE}.
- EDIT.
- WHEN *fieldname* EQ an embedded quote in a literal.
- DEFINE commands.
- DEFINE attributes in Master Files.
- Database Administrator (DBA) attributes in Master Files (for example, VALUE = *fieldname* EQ an embedded quote in a literal).
- ACCEPT=, DESCRIPTION=, TITLE= attributes in Master Files.
- AS.
- DECODE.

Example Specifying the Data Value O'BRIEN in a Quote-Delimited Literal String

The following example illustrates the use of quotation marks for the correct interpretation of the data value O'BRIEN:

```
TABLE FILE VIDEOTRK
PRINT LASTNAME
WHERE LASTNAME IS 'O''BRIEN'
END
```

Concatenating Character Strings

You can write an expression that concatenates two or more alphanumeric constants and/or fields into a single character string. The concatenation operator has two forms, as shown in the following table:

Symbol	Represents	Description
	Weak concatenation	Preserves trailing blanks.
	Strong concatenation	Moves trailing blanks to end of concatenated string.

Example Concatenating Character Strings

The following example uses the EDIT function to extract the first initial from a first name. It then uses both strong and weak concatenation to produce the last name, followed by a comma, followed by the first initial, followed by a period:

```
DEFINE FILE EMPLOYEE
FIRST_INIT/A1 = EDIT(FIRST_NAME, '9$$$$$$$$$');
NAME/A19 = LAST_NAME ||(', ' | FIRST_INIT |'.');
END
TABLE FILE EMPLOYEE
PRINT NAME WHERE LAST_NAME IS 'BANNING'
END
```

The output is:

```
NAME
----
BANNING, J.
```

The request evaluates the expressions as follows:

1. The EDIT function extracts the initial J from FIRST_NAME.
2. The expression in parentheses returns the value:
, J.
3. LAST_NAME is concatenated to the string derived in step 2 to produce:
Banning, J.

While LAST_NAME has the format A15 in the EMPLOYEE Master File, strong concatenation suppresses the trailing blanks. Regardless of the suppression or inclusion of blanks, the resulting field name, NAME, has a length of 19 characters (A19).

Creating a Logical Expression

A logical expression determines whether a particular condition is true. There are two kinds of logical expressions: relational and Boolean. The entities to be compared determine the kind of expression used:

- A relational expression returns TRUE or FALSE based on a comparison of two individual values (either field values or constants).
- A Boolean expression returns TRUE or FALSE based on the outcome of two or more relational expressions.

You can use a logical expression to assign a value to a numeric field. If the expression is true, the field receives the value 1. If the expression is false, the field receives the value 0.

Reference

Logical Operators

The following is a list of common operators used in logical expressions. For information on relational operators and additional operators available for record selection using WHERE and IF, see Chapter 5, *Selecting Records for Your Report*.

Operator	Description
EQ	Returns the value TRUE if the value on the left is equal to the value on the right.
NE	Returns the value TRUE if the value on the left is not equal to the value on the right.
GE	Returns the value TRUE if the value on the left is greater than or equal to the value on the right.
GT	Returns the value TRUE if the value on the left is greater than the value on the right.
LE	Returns the value TRUE if the value on the left is less than or equal to the value on the right.
LT	Returns the value TRUE if the value on the left is less than the value on the right.
AND	Returns the value TRUE if both operands are true.
OR	Returns the value TRUE if either operand is true.
NOT	Returns the value TRUE if the operand is false.
CONTAINS	Contains the specified character strings.
OMITS	Omits the specified character strings.

Syntax

How to Write a Relational Expression

Any of the following are valid for a relational expression

value {EQ|NE} *value*

value {GE|GT} *value*

value {LE|LT} *value*

character_value {CONTAINS|OMITS} *character_value*

where:

value

Is a field value or constant.

character_value

Is a character string. If it contains blanks, the string must be enclosed in single quotation marks.

Syntax

How to Write a Boolean Expression

Either of the following are valid for a Boolean expression

(*relational_expression*) {AND|OR} (*relational_expression*)

NOT (*logical_expression*)

where:

relational_expression

Is an expression based on a comparison of two individual values (either field values or constants).

logical_expression

Is an expression that evaluates to the value TRUE or FALSE. If the expression is true, the field receives the value 1. If the expression is false, the field receives the value 0. The expression must be enclosed in parentheses.

Creating a Conditional Expression

A conditional expression assigns a value based on the result of a logical expression. The assigned value can be numeric or alphanumeric. A conditional expression can include up to 16 IF criteria.

Note: Unlike selection criteria using IF, all alphanumeric values in conditional expressions must be enclosed in single quotation marks. For example, IF COUNTRY EQ 'ENGLAND'.

Syntax

How to Write a Conditional Expression

```
IF expression1 THEN expression2 [ELSE expression3]
```

where:

expression1

Is the expression that is evaluated to determine whether the field is assigned the value of *expression2* or of *expression3*.

expression2

Is an expression that results in a format compatible with the format assigned to the field. It may be a conditional expression, in which case you must enclose it in parentheses.

expression3

Is an expression that results in a format compatible with the format assigned to the field. Enclosure of the expression in parentheses is optional.

ELSE

Is optional, along with *expression3*. However, if you do not specify an ELSE condition and the IF condition is not met, the value is taken from the last evaluated condition.

Note that the final sorted report may display mixed values. This depends on whether a DEFINE or a COMPUTE is used, and if a data record is evaluated before or after aggregation.

Note: The expressions following THEN and ELSE must result in a format that is compatible with the format assigned to the field. Each of these expressions may itself be a conditional expression. However, the expression following IF may not be an IF ... THEN ... ELSE expression (for example, IF ... IF ...).

Example**Supplying a Value With a Conditional Expression**

The following example uses a conditional expression to assign the value NONE to the field BANK_NAME when it is missing a data value (that is, when the field has no data in the data source):

```
DEFINE FILE EMPLOYEE
BANK_NAME/A20 = IF BANK_NAME EQ ' ' THEN 'NONE'
ELSE BANK_NAME;
END

TABLE FILE EMPLOYEE
PRINT CURR_SAL AND BANK_NAME
BY EMP_ID BY BANK_ACCT
END
```

The output is:

```
PAGE      1
```

EMP_ID	BANK_ACCT	CURR_SAL	BANK_NAME
-----	-----	-----	-----
071382660		\$11,000.00	NONE
112847612		\$13,200.00	NONE
117593129	40950036	\$18,480.00	STATE
119265415		\$9,500.00	NONE
119329144	160633	\$29,700.00	BEST BANK
123764317	819000702	\$26,862.00	ASSOCIATED
126724188		\$21,120.00	NONE
219984371		\$18,480.00	NONE
326179357	122850108	\$21,780.00	ASSOCIATED
451123478	136500120	\$16,100.00	ASSOCIATED
543729165		\$9,000.00	NONE
818692173	163800144	\$27,062.00	BANK ASSOCIATION

Example

Defining a True or False Condition

You can also define a true or false condition and then test it to control report output. The following example assigns the value TRUE to the field MYTEST if either of the relational expressions in parentheses is true. It then tests the value of MYTEST:

```
DEFINE FILE EMPLOYEE  
MYTEST= (CURR_SAL GE 11000) OR (DEPARTMENT EQ 'MIS');  
END
```

```
TABLE FILE EMPLOYEE  
PRINT CURR_SAL AND DEPARTMENT  
BY EMP_ID  
IF MYTEST IS TRUE  
END
```

The output is:

```
PAGE      1  
  
EMP_ID      CURR_SAL  DEPARTMENT  
-----  
071382660    $11,000.00  PRODUCTION  
112847612    $13,200.00   MIS  
117593129    $18,480.00   MIS  
119329144    $29,700.00  PRODUCTION  
123764317    $26,862.00  PRODUCTION  
126724188    $21,120.00  PRODUCTION  
219984371    $18,480.00   MIS  
326179357    $21,780.00   MIS  
451123478    $16,100.00  PRODUCTION  
543729165     $9,000.00   MIS  
818692173    $27,062.00   MIS
```

Note: Testing for a TRUE or FALSE condition is valid only with the IF command. It is not valid with WHERE.

CHAPTER 9

Customizing Tabular Reports

Topics:

- Creating Paging and Numbering
- Separating Sections of a Report: SKIP-LINE and UNDER-LINE
- Suppressing Fields: SUP-PRINT or NOPRINT
- Creating New Column Titles: AS
- Customizing Column Names: SET QUALTITLES
- Positioning Columns: IN
- Reducing a Report's Width: FOLD-LINE and OVER
- Controlling Column Spacing: SET SPACES
- Column Title Justification
- Customizing Reports With SET Parameters
- Producing Headings and Footings
- Conditionally Formatting Reports With the WHEN Clause
- Controlling the Display of Empty Reports

FOCUS provides a variety of formatting options that enable you to customize your reports. For example, you can specify page breaks, rename report column titles, and add subfoot text to the bottom of pages.

Note: FOCUS formats reports automatically using defaults based on the formats of fields. However, you can override these defaults to customize your report format to suit your individual requirements.

Creating Paging and Numbering

The appearance of your report can be enhanced by controlling paging and page numbering. You can:

- Specify a page break (PAGE-BREAK).
- Reposition page numbers (TABPAGENO).
- Suppress page numbers (SET PAGE).
- Prevent an undesirable split (NOSPLIT).

Specifying a Page Break: PAGE-BREAK

Use the PAGE-BREAK option to start a new page each time the specified sort field value changes or to prevent information that should be grouped together from being presented over more than one page.

To specify a page break, use PAGE-BREAK in either an ON phrase or BY phrase immediately after the sort field on which you want to break the page. You can also use PAGE-BREAK to:

- Reset the report page to 1 at specified points (REPAGE).
- Specify conditional page breaks in the printing of a report (with WHEN).

Syntax

How to Specify a Page Break

The syntax is

```
{ON|BY} fieldname PAGE-BREAK [REPAGE][WHEN expression;
```

where:

fieldname

Is a sort field. A change in the sort field value causes a page-break.

REPAGE

Resets the page number to 1 at the sort break or, if WHEN is used, whenever the conditions in the WHEN clause are met.

WHEN *expression*

Specifies conditional page breaks in the printing of a report as determined by a Boolean expression (see *Using Direct Operators in Headings and Footings* on page 9-44).

Reference**Usage Notes for Page Breaks**

- Page headings and column titles will appear at the top of each new page.
- Put the PAGE-BREAK command on the lowest-level sort field at which the page break is to occur.
- Page breaks automatically occur whenever a higher-level sort field changes.
- PAGE-BREAK is ignored when report output is stored in HOLD, SAVE, or SAVB files (see Chapter 11, *Saving and Reusing Report Output*).

Example**Specifying a Page Break**

For example:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID
BY SALARY IN-GROUPS-OF 5000
BY PCT_INC BY DAT_INC
ON SALARY PAGE-BREAK
END
```

The first two pages of this report display as:

PAGE 1

SALARY	PCT_INC	DAT_INC	EMP_ID
-----	-----	-----	-----
\$5,000.00	.00	82/01/04	119265415
		82/04/01	543729165
	.04	82/06/11	543729165
	.05	82/05/14	119265415

PAGE 2

SALARY	PCT_INC	DAT_INC	EMP_ID
-----	-----	-----	-----
\$10,000.00	.10	82/01/01	071382660
			112847612
	.12	81/01/01	071382660

Inserting Page Numbers: TABPAGENO

By default, FOCUS reserves the first two lines of each report page: the first line contains the page number at the left margin—that is, in the top-left corner of the page—and the following line is blank. You can change the position of the page number with the TABPAGENO system variable.

TABPAGENO contains the page number of the current page and acts like a field name. Therefore, it can be positioned in a heading or footing (or subhead/subfoot). The default page number in the top left-hand corner is automatically suppressed when this variable is used.

Example Inserting Page Numbers

For example, this request

```
TABLE FILE PROD
"<TABPAGENO"
PRINT PACKAGE AND UNIT_COST
BY PROD_NAME BY PROD_CODE
ON PROD_NAME PAGE-BREAK
END
```

creates the following report (of which the first two pages are shown):

1			
PROD_NAME	PROD_CODE	PACKAGE	UNIT_COST
-----	-----	-----	-----
AMERICAN CHEESE	C7	8 OUNCES	\$2.19

2			
PROD_NAME	PROD_CODE	PACKAGE	UNIT_COST
-----	-----	-----	-----
BUTTER MILK	C14	32 OUNCES	\$1.89

Note that FOCUS continues to reserve the top two lines of every report page.

Suppressing Page Numbers: SET PAGE

Automatic page numbering can also be suppressed with the SET PAGE command.

Syntax

How to Suppress Page Numbers

To suppress page numbering, the syntax is

```
SET PAGE = {OFF|NOPAGE|TOP}
```

where:

OFF

Suppresses automatic page numbering. You can still use the variable TABPAGENO as described in *Inserting Page Numbers: TABPAGENO* on page 9-4. Note that FOCUS reserves the top two lines of every page.

NOPAGE

Suppresses all page indicators and makes the first two lines of each report page available for your use. NOPAGE does not issue page ejects; they are issued if you use SET PAGE=OFF.

TOP

Omits the line at the top of each page of the report output for the page number and the blank line that follows it. The first line of the report output contains the heading, if one was specified, or the column titles, if there is no heading.

Preventing an Undesirable Split

Page breaks sometimes occur where report information has been logically grouped by sort field(s), causing one or two lines to appear by themselves on the next page or screen. To prevent this, use NOSPLIT in either an ON phrase or immediately after the first reference to the sort field in a BY phrase.

Syntax

How to Prevent an Undesirable Split

The syntax is

```
{ON|BY} fieldname NOSPLIT
```

where:

fieldname

Is the name of the sort field for which sort groups will be kept together on the same page.

Whenever the value of the specified field changes, FOCUS determines if the total number of lines related to the new value can fit on the current page. If they cannot, the page breaks and the group of lines appears on the next page.

Reference

Usage Notes for Preventing an Undesirable Split

- Only one NOSPLIT option is allowed per report. If a PAGE-BREAK option also exists in the request, it must relate to a higher-level sort field; otherwise, NOSPLIT will be ignored.
- Subtotals, footings, subheads, and subfoots are placed on the same page as the detail lines; headings are placed on the new page.
- NOSPLIT is ignored when report output is stored in HOLD, SAVE, or SAVB files (see Chapter 11, *Saving and Reusing Report Output*).
- NOSPLIT is not compatible with the TABLEF command and produces an FOC037 error message.

Example

Preventing an Undesirable Split

```
TABLE FILE EMPLOYEE
PRINT DED_CODE AND DED_AMT
BY PAY_DATE BY LAST_NAME
ON LAST_NAME NOSPLIT
END
```

Depending upon how many lines your output device is set to, the first two pages of the previous request might display as:

PAGE1

PAY_DATE	LAST_NAME	DED_CODE	DED_AMT
81/11/30	CROSS	CITY	\$7.52
		FED	\$638.96
		FICA	\$526.20
		HLTH	\$32.22
		LIFE	\$19.33
		SAVE	\$77.32
		STAT	\$105.24
	STEVENS	CITY	\$.83
		FED	\$70.83
		FICA	\$58.33
		STAT	\$11.67

PAGE2

PAY_DATE	LAST_NAME	DED_CODE	DED_AMT
81/12/31	CROSS	CITY	\$7.52
		FED	\$638.96
		FICA	\$526.20
		HLTH	\$32.22
		LIFE	\$19.33
		SAVE	\$77.32
		STAT	\$105.24
	STEVENS	CITY	\$.83
		FED	\$70.83
		FICA	\$58.33
		STAT	\$11.67

Here are the first two pages without NOSPLIT:

PAGE 1

PAY_DATE	LAST_NAME	DED_CODE	DED_AMT
81/11/30	CROSS	CITY	\$7.52
		FED	\$638.96
		FICA	\$526.20
		HLTH	\$32.22
		LIFE	\$19.33
		SAVE	\$77.32
		STAT	\$105.24
	STEVENS	CITY	\$.83
		FED	\$70.83
		FICA	\$58.33
		STAT	\$11.67
81/12/31	CROSS	CITY	\$7.52
		FED	\$638.96
		FICA	\$526.20
		HLTH	\$32.22
		LIFE	\$19.33
		SAVE	\$77.32

PAGE 2

PAY_DATE	LAST_NAME	DED_CODE	DED_AMT
81/12/31	CROSS	STAT	\$105.24
	STEVENS	CITY	\$.83
		FED	\$70.83
		FICA	\$58.33
		STAT	\$11.67
82/01/29	CROSS	CITY	\$7.52
		FED	\$638.96
		FICA	\$526.20
		HLTH	\$32.22
		LIFE	\$19.33
		SAVE	\$77.32
		STAT	\$105.24
	IRVING	CITY	\$6.10
		FED	\$518.92
		FICA	\$427.35
		HLTH	\$50.87
		LIFE	\$30.52

The report without NOSPLIT has an undesirable split for Cross on Page 2, whereas the report using NOSPLIT does not.

Separating Sections of a Report: SKIP-LINE and UNDER-LINE

To make a detailed report easier to read and interpret, you can separate sections of it—individual lines, or entire sort groups—by inserting blank lines between them, or (for sort groups only) by underlining them.

Adding Blank Lines: SKIP-LINE

Report information often stands out more clearly if lines are skipped between individual lines, or between sort groups. You can use SKIP-LINE with either a sort field or a display field.

- If you use SKIP-LINE with a sort field, FOCUS inserts a blank line between each section of the report.
- If you use SKIP-LINE with a display field, FOCUS inserts a blank line between each line of the report—in effect, double-spacing the report. Double spacing is especially helpful when a report will be used as a review document, as it makes it easy for the reader to write comments next to individual lines.

Syntax

How to Add Blank Lines

To add blank lines, use SKIP-LINE with the keyword ON, or BY. Use the WHEN clause to specify conditional blank lines in the printing of a report. The syntax is

```
display fieldname SKIP-LINE  
{ON|BY} fieldname SKIP-LINE [WHEN expression;]
```

where:

display

Is any display command.

fieldname

Is used so that when the value of this field changes, a blank line is inserted before the next set of values.

WHEN expression

Specifies conditional blank lines in the printing of a report as determined by a Boolean expression (see *Using Direct Operators in Headings and Footings* on page 9-44).

You can use only one SKIP-LINE in each report request. You do not have to enter it on its own line; instead, include it after the field name or sort field for which you want to insert a blank line.

Reference

Usage Notes for Adding Blank Lines

Keep the following in mind when using SKIP-LINE:

- If the field name is a sort field, a blank line is inserted just before every change in value of the sort field.
- If the field name is a display field, a blank line is inserted after every printed line. The WHEN clause does not apply to display fields.
- This is one of the only ON conditions that does not have to refer solely to sort control (BY) fields.
- Only one SKIP-LINE option is allowed per request and it may affect more than one sort field.

Example

Adding Blank Lines

For example:

```
DEFINE FILE EMPLOYEE
INCREASE/D8.2M = .05*CURR_SAL;
CURR_SAL/D8.2M=CURR_SAL;
NEWSAL/D8.2M=CURR_SAL + INCREASE;
END

TABLE FILE EMPLOYEE
PRINT CURR_SAL OVER INCREASE OVER NEWSAL
BY EMP_ID BY LAST_NAME BY FIRST_NAME
ON EMP_ID SKIP-LINE
END
```

The first part of the report output is shown below:

PAGE 1

EMP_ID	LAST_NAME	FIRST_NAME		
-----	-----	-----		
071382660	STEVENS	ALFRED	CURR_SAL	\$11,000.00
			INCREASE	\$550.00
			NEWSAL	\$11,550.00
112847612	SMITH	MARY	CURR_SAL	\$13,200.00
			INCREASE	\$660.00
			NEWSAL	\$13,860.00
117593129	JONES	DIANE	CURR_SAL	\$18,480.00
			INCREASE	\$924.00
			NEWSAL	\$19,404.00
119265415	SMITH	RICHARD	CURR_SAL	\$9,500.00
			INCREASE	\$475.00
			NEWSAL	\$9,975.00

Underlining Values: UNDER-LINE

Drawing a line across the page after all of the information for a particular section has been displayed can enhance the readability of a printed report.

Syntax How to Underline Values

The syntax is

```
{ON|BY} fieldname UNDER-LINE [WHEN expression;
```

where:

fieldname

Is used so that when the value of the field changes, a line is drawn. A line is automatically drawn after any other option such as RECAP or SUB-TOTAL (but before PAGE-BREAK).

WHEN *expression*

Specifies conditional underlines in the printing of a report, as determined by a Boolean expression (see *Inserting Page Numbers: TABPAGENO* on page 9-4).

Example Underlining Values

For example:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND BANK_ACCT AND LAST_NAME
BY BANK_NAME
ON BANK_NAME UNDER-LINE
END
```

The request produces the following report:

PAGE 1			
BANK_NAME	EMP_ID	BANK_ACCT	LAST_NAME
	071382660		STEVENS
	112847612		SMITH
	119265415		SMITH
	126724188		ROMANS
	219984371		MCCOY
	543729165		GREENSPAN
ASSOCIATED	123764317	819000702	IRVING
	326179357	122850108	BLACKWOOD
	451123478	136500120	MCKNIGHT
BANK ASSOCIATION	818692173	163800144	CROSS
BEST BANK	119329144	160633	BANNING
STATE	117593129	40950036	JONES

Suppressing Fields: SUP-PRINT or NOPRINT

You can create reports that do not display the values or titles of fields, but only use those fields to produce specific effects. FOCUS provides options to suppress the printing of field values: NOPRINT and SUP-PRINT.

Syntax

How to Suppress Fields

The syntax is:

```
display fieldname {SUP-PRINT|NOPRINT}  
{ON|BY} fieldname {SUP-PRINT|NOPRINT}
```

Valid values are:

display

Is any display command.

fieldname

Is a sort field or display field. The values of the field may be used, but they will not be displayed.

Reference

Usage Notes for Suppressing Fields

- If you put a NOPRINT or SUP-PRINT phrase in a computed field, you must then repeat AND COMPUTE before the next computed field.
- If you use the NOPRINT option with a BY field and create a HOLD file, the BY field is excluded from the file. For example, a request that includes the phrase

```
BY DEPARTMENT NOPRINT
```

will result in a HOLD file that does not contain the DEPARTMENT field.

Example

Suppressing Fields

For example, to print a list of employee names in alphabetical order, if you simply used the request

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
END
```

you would get a report that lists the last names of employees in the order they were entered into the data source:

PAGE 1

LAST_NAME

STEVENS
SMITH
JONES
SMITH
BANNING
IRVING
ROMANS
MCCOY
BLACKWOOD
MCKNIGHT
GREENSPAN
CROSS

To print the last names in alphabetical order, use NOPRINT in conjunction with a BY phrase

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
BY LAST_NAME NOPRINT
END
```

which produces the desired result:

PAGE 1

LAST_NAME

BANNING
BLACKWOOD
CROSS
GREENSPAN
IRVING
JONES
MCCOY
MCKNIGHT
ROMANS
SMITH
SMITH
STEVENS

Example

Suppressing a Sort Field

Consider the following example, where the report is sorted, but the field that determines the sort order is not displayed:

```
TABLE FILE SALES
PRINT UNIT_SOLD AND DELIVER_AMT
BY CITY BY PROD_CODE BY RETAIL_PRICE
ON RETAIL_PRICE SUP-PRINT
END
```

The report is as follows:

PAGE 1

CITY	PROD_CODE	UNIT_SOLD	DELIVER_AMT
----	-----	-----	-----
NEW YORK	B10	30	30
	B17	20	40
	B20	15	30
	C17	12	10
	D12	20	30
	E1	30	25
	E3	35	25
NEWARK	B10	13	30
	B12	29	30
STAMFORD	B10	60	80
	B12	40	20
	B17	29	30
	C13	25	30
	C7	45	50
	D12	27	40
	E2	80	100
	E3	70	80
UNIONDALE	B20	25	40
	C7	40	40

Also, consider the following example

```
TABLE FILE CAR
SUM SALES BY COUNTRY
BY CAR
ON COUNTRY SUB-TOTAL SUP-PRINT PAGE-BREAK
END
```

which generates this report (only the beginning is shown):

PAGE 1

CAR	SALES
---	-----
JAGUAR	12000
JENSEN	0
TRIUMPH	0
*TOTAL ENGLAND	12000

MORE

PAGE 2

CAR	SALES
---	-----
PEUGEOT	0
*TOTAL FRANCE	0

MORE

PAGE 3

CAR	SALES
---	-----
ALFA ROMEO	30200
MASERATI	0
*TOTAL ITALY	30200

Creating New Column Titles: AS

Use the AS option to rename existing column titles in your reports. Any of the following titles can be changed with an AS phrase:

- ACROSS titles can be replaced by one line of text only.
- A SUBTOTAL line can be replaced by one line of text only.
- FOR phrases.
- Fields for the MATCH command.

Syntax

How to Create Column Titles

The syntax for changing default titles is

```
field AS 'title1, title2,...'
```

where:

field

Can be a sort field, display field, column total, or row total.

title

Is the new column title enclosed in single quotation marks.

To specify multiple lines in a column title, separate each line's text with commas. Up to five lines are allowed.

Reference

Usage Notes for New Column Titles

- When using FOLD-LINE, the titles appear one over the other. No more than one line per title is allowed with FOLD-LINE. (See *Reducing a Report's Width: FOLD-LINE and OVER* on page 9-22.)
- The use of a title line larger than the format size of the data is one convenient way to space out a report across the columns of the page. For instance,

```
PRINT UNITS BY MONTH AS ' MONTH'
```

shifts the title for MONTH to the right and all other columns, in this case UNITS, shift to the right. For more information on changing the column position, see *Reducing a Report's Width: FOLD-LINE and OVER* on page 9-22.
- If you do not want any field name or title displayed in the report, you can also use the AS phrase by entering two consecutive single quotation marks. For example:

```
PRINT LAST_NAME AS ''
```

To display underscores, enclose blanks in single quotation marks.
- If you put an AS phrase in a computed field, you must then repeat the keyword COMPUTE before the next computed field.

Example

Creating New Column Titles

For example:

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AS 'NAME' AND LAST_NAME AS ''
BY DEPARTMENT
BY EMP_ID AS 'EMPLOYEE,NUMBER'
END
```

This request produces the following report:

PAGE 1

DEPARTMENT	EMPLOYEE NUMBER	NAME	
MIS	112847612	MARY	SMITH
	117593129	DIANE	JONES
	219984371	JOHN	MCCOY
	326179357	ROSEMARIE	BLACKWOOD
	543729165	MARY	GREENSPAN
PRODUCTION	818692173	BARBARA	CROSS
	071382660	ALFRED	STEVENS
	119265415	RICHARD	SMITH
	119329144	JOHN	BANNING
	123764317	JOAN	IRVING
	126724188	ANTHONY	ROMANS
	451123478	ROGER	MCKNIGHT

Customizing Column Names: SET QUALTITLES

The FOCUS SET command, SET QUALTITLES, enables you to determine whether or not duplicate field names appear as qualified column titles in TABLE output. (For more information about long and qualified field names, see Chapter 2, *Displaying Report Data*.)

Syntax

How to Customize Column Headings

The syntax is

```
SET QUALTITLES = {ON|OFF}
```

where:

ON

Enables qualified column titles when duplicate field names exist and FIELDNAME is set to NEW.

OFF

Disables qualified column titles. OFF is the default value.

SET QUALTITLES may also be issued from within a TABLE request.

Qualified column titles are automatically used, even if qualified field names are not used in the request.

Reference

Usage Notes for Qualified Column Titles

- AS names are used if duplicate field names are referenced in a MATCH request.
- AS names are used when duplicate field names exist in a HOLD file.

Positioning Columns: IN

FOCUS automatically formats a page and uses common default values for determining column positions and spacing. You can override these defaults by specifying the absolute or relative column position where a data value is to appear on a report.

Syntax

How to Position Columns

The syntax is

```
field IN {n|+n}
```

Valid values are:

field

Is the field (that is, the column) that you want to move.

n

Is a number indicating the absolute position of the column.

+*n*

Is a number indicating the relative position of the column. That is, +*n* is the number of characters to the right of the last column.

Reference

Usage Notes for Positioning Columns

- The IN phrase can be used with ACROSS to specify both the starting column of the entire ACROSS set as well as the spacing between each column within the ACROSS.
- When one field is positioned over another (for example, when OVER or FOLD-LINE is used; see *Reducing a Report's Width: FOLD-LINE and OVER* on page 9-22), the positions apply to the line on which the referenced field occurs.

Example

Positioning Columns

For example:

```
TABLE FILE EMPLOYEE
PRINT BANK_NAME IN 1
BY HIGHEST BANK_ACCT IN 26
BY LAST_NAME IN 40
END
```

This request produces the following report. There is a blank line following SMITH because LAST_NAME is a sort field and there are two employees named Smith in the database.

PAGE 1

BANK_NAME	BANK_ACCT	LAST_NAME
ASSOCIATED	819000702	IRVING
BANK ASSOCIATION	163800144	CROSS
ASSOCIATED	136500120	MCKNIGHT
ASSOCIATED	122850108	BLACKWOOD
STATE	40950036	JONES
BEST BANK	160633	BANNING
		GREENSPAN
		MCCOY
		ROMANS
		SMITH
		STEVENS

Example

Positioning Columns With ACROSS

The IN phrase can be used with ACROSS to specify both the starting column of the entire ACROSS set as well as the spacing between each column within the ACROSS, as shown in the following example:

```
TABLE FILE CAR
SUM UNITS IN +1 ACROSS CAR IN 30
BY COUNTRY
END
```

This will place one extra space between the data columns in the matrix and display the ACROSS sets beginning in Position 30, as shown in the first page of the report below.

PAGE	1			
		CAR		
		ALFA ROMEO	AUDI	BMW
COUNTRY				
ENGLAND		.	.	.
FRANCE		.	.	.
ITALY		30200	.	.
JAPAN		.	.	.
W GERMANY		.	7800	80390

Example **Positioning Columns With FOLD-LINE**

When one field is positioned over another (for example, when OVER or FOLD-LINE is used, see *Reducing a Report's Width: FOLD-LINE and OVER* on page 9-22), the positions apply to the line on which the referenced field occurs, as in the following example:

```
TABLE FILE CAR
SUM RCOST BY CAR
BY COUNTRY IN 25
ON COUNTRY FOLD-LINE
END
```

which creates this report, in which COUNTRY starts in column 25 and RCOST appears on the second line.

PAGE 1

CAR		COUNTRY
---		-----
	RETAIL_COST	

ALFA ROMEO		ITALY
	19,565	
AUDI		W GERMANY
	5,970	
BMW		W GERMANY
	58,762	
DATSUN		JAPAN
	3,139	
JAGUAR		ENGLAND
	22,369	
JENSEN		ENGLAND
	17,850	
MASERATI		ITALY
	31,500	
PEUGEOT		FRANCE
	5,610	

Example Positioning Columns With OVER

Consider the following report request

```
TABLE FILE CAR
PRINT SALES IN 50 OVER RCOST IN 50
BY COUNTRY IN 10 BY MODEL
END
```

which generates this report:

PAGE	1		
	COUNTRY	MODEL	
	-----	-----	
	ENGLAND	INTERCEPTOR III	SALES 0
			RETAIL_COST 17,850
		TR7	SALES 0
			RETAIL_COST 5,100
		U12XKE AUTO	SALES 0
			RETAIL_COST 8,878
		XJ12L AUTO	SALES 12000
			RETAIL_COST 13,491
	FRANCE	504 4 DOOR	SALES 0
			RETAIL_COST 5,610
	ITALY	DORA 2 DOOR	SALES 0
			RETAIL_COST 31,500
		2000 GT VELOCE	SALES 12400
			RETAIL_COST 6,820
		2000 SPIDER VELOCE	SALES 13000
			RETAIL_COST 6,820
		2000 4 DOOR BERLINA	SALES 4800

Reducing a Report's Width: FOLD-LINE and OVER

Wide reports are difficult to read, especially on a screen. To reduce a report's width, use FOLD-LINE and OVER.

Compressing the Columns of Reports: FOLD-LINE

A single line on a report can be folded to compress it into fewer columns. This enables you to display a wide report on a narrow screen and enhance the appearance of many reports which might otherwise have wasted space under sort control fields which change infrequently.

Syntax

How to Compress Report Columns

The syntax for specifying the point of line fold is

```
display fieldname ... FOLD-LINE fieldname ...  
{ON|BY} fieldname FOLD-LINE
```

where:

display

Is any display command.

fieldname

Causes columns to be placed on a separate line when the value of the field changes in the BY or ON phrase. The field name may be a sort field or display field. When it is a display field, it will be placed under the preceding field.

Reference

Usage Notes for Compressing Report Columns

- The second half of the folded line is offset by two spaces from the first part when the line is folded on a sort control field.
- Instead of FOLD-LINE, you can also use the OVER phrase to decrease the width of reports, described in *Decreasing the Width of a Report: OVER* on page 9-23.
- When the point of line folding is after a display field, there is no offset. A simple way to change the line alignment is to use a title with leading blanks. (See *Creating New Column Titles: AS* on page 9-15.)

Example

Compressing Report Columns

For example:

```
TABLE FILE EMPLOYEE
SUM ED_HRS BY DEPARTMENT
PRINT ED_HRS AND LAST_NAME AND FIRST_NAME
BY DEPARTMENT BY HIGHEST BANK_ACCT
ON DEPARTMENT FOLD-LINE
END
```

This request produces the following report.

PAGE 1

DEPARTMENT

ED_HRS	BANK_ACCT	ED_HRS	LAST_NAME	FIRST_NAME
MIS				
231.00	163800144	45.00	CROSS	BARBARA
	122850108	75.00	BLACKWOOD	ROSEMARIE
	40950036	50.00	JONES	DIANE
		36.00	SMITH	MARY
		.00	MCCOY	JOHN
		25.00	GREENSPAN	MARY
PRODUCTION				
120.00	819000702	30.00	IRVING	JOAN
	136500120	50.00	MCKNIGHT	ROGER
	160633	.00	BANNING	JOHN
		25.00	STEVENS	ALFRED
		10.00	SMITH	RICHARD
		5.00	ROMANS	ANTHONY

Decreasing the Width of a Report: OVER

One way to decrease the width of your report (particularly when using the ACROSS phrase) is to use OVER. OVER places field names over one another. The syntax is

```
display fieldname1 OVER fieldname2 OVER fieldname3 ...
```

where:

display

Is any display command.

fieldname

Causes the fields listed to be placed over each other, instead of printed beside each other in a row. The field names must be a display field.

Reference Usage Notes for Decreasing Report Width

Keep the following in mind when using OVER:

- For more complex combinations of IN and OVER, you may want to create subfoots with data. Subfoots with data are discussed in *Producing Headings and Footings* on page 9-29.
- Text fields cannot be specified with OVER.

Example Decreasing the Width of a Report

For example:

```
TABLE FILE EMPLOYEE
SUM GROSS OVER DED_AMT OVER
COMPUTE NET/D8.2M = GROSS - DED_AMT;
ACROSS DEPARTMENT
END
```

The request produces the following report. Notice the ACROSS values display to the left, not directly above the data values.

PAGE 1

	DEPARTMENT	
	MIS	PRODUCTION
GROSS	\$50,499.12	\$50,922.38
DED_AMT	\$28,187.25	\$23,391.35
NET	\$22,311.88	\$27,531.03

Without the OVER phrase, the report would look like this:

PAGE 1						
DEPARTMENT MIS			PRODUCTION			
GROSS	DED_AMT	NET	GROSS	DED_AMT	NET	
\$50,499.12	\$28,187.25	\$22,311.88	\$50,922.38	\$23,391.35	\$27,531.03	

Controlling Column Spacing: SET SPACES

By default, FOCUS puts one or two spaces between report columns, depending on the output width. The SET SPACES command enables you to control the number of spaces between columns in a report.

Syntax

How to Control Column Spacing

The syntax is:

```
SET SPACES = {n|AUTO}
```

Valid values are:

n

Is a number indicating from 1 to 8 spaces.

AUTO

Specifies that FOCUS automatically puts one or two spaces between columns depending on report output and available output length. AUTO is the default setting.

SET SPACES may also be issued from within a TABLE request.

For ACROSS phrases, SET SPACES *n* controls the distance between ACROSS sets. Within an ACROSS set, the distance between fields is always one space and cannot be changed.

Example

Controlling Column Spacing

The following example illustrates the use of ACROSS with SET SPACES:

```
TABLE FILE CAR
SUM DEALER_COST RETAIL_COST ACROSS CAR BY COUNTRY
IF CAR EQ 'ALFA ROMEO' OR 'BMW'
ON TABLE SET SPACES 8
END
```

The ACROSS set consists of the fields DEALER_COST and RETAIL_COST. The distance between each set is eight spaces.

PAGE 1

COUNTRY	CAR ALFA ROMEO DEALER_COST RETAIL_COST	BMW DEALER_COST RETAIL_COST
ITALY	16,235 19,565	. .
W GERMANY	. .	49,500 58,762

Column Title Justification

You can specify whether column titles in a report are left justified, right justified, or centered. By default, column titles for alphanumeric fields are left justified, and column titles for numeric and date fields are right justified over the displayed column.

Syntax

How to Justify Column Titles

The syntax to alter default justification is

```
fieldname [alignment] [/format]
```

where:

alignment

Specifies the alignment of the column title.

/R specifies that the column title is to be right justified.

/L specifies that the column title is to be left justified.

/C specifies that the column title is to be centered.

/format

Is an optional format specification for the field.

Reference

Usage Notes for Justifying Column Titles

- You may specify justification for display fields, BY fields, ACROSS fields, column totals, and row totals (see Chapter 2, *Displaying Report Data*). For ACROSS fields, data values, not column titles, are justified as specified.
- For display commands and row totals only, the justification parameter may be combined with a format specification, which precedes or follows the justification parameter (for example, PRINT CAR/A8/R MODEL/C/A15).
- If a title is specified with an AS phrase or in the Master File, that title will be justified as specified for the field.
- When multiple ACROSS fields are requested, justification is performed on the lowest ACROSS level only. All other justification parameters for ACROSS fields are ignored.

Example

Justifying Column Titles

The following example illustrates column title justification with a format specification, a BY field specification, and an AS phrase specification:

```
TABLE FILE CAR
PRINT MODEL/A10 STANDARD/A15/R AS 'RJUST,STANDARD' BY CAR/C
WHERE CAR EQ 'JAGUAR' OR 'TOYOTA'
END
```

PAGE 1

CAR	MODEL	RJUST STANDARD
JAGUAR	V12XKE AUT XJ12L AUTO	POWER STEERING RECLINING BUCKE WHITEWALL RADIA WRAP AROUND BUM 4 WHEEL DISC BR
TOYOTA	COROLLA 4	BODY SIDE MOLDI MACPHERSON STRU

Customizing Reports With SET Parameters

Most SET commands that change system defaults may be issued from within a report request. Many SET command parameters can be used to enhance the readability and usefulness of your reports. The SET command, when used in this manner, affects only the request in which it occurs. For a complete list of SET parameters and acceptable values, see the *Developing Applications* manual.

Syntax

How to Use SET Parameters in Requests

The syntax is

```
ON TABLE SET parameter value [AND parameter value...]
```

where:

parameter

Is the SET command parameter that you wish to change.

value

Replaces the default value.

Example **Setting Parameters in a Report Request**

For example, this request changes the NODATA character for missing data from a period (default) to the word NONE. No equal sign is allowed.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY LAST_NAME BY FIRST_NAME
ACROSS DEPARTMENT
ON TABLE SET NODATA NONE
END
```

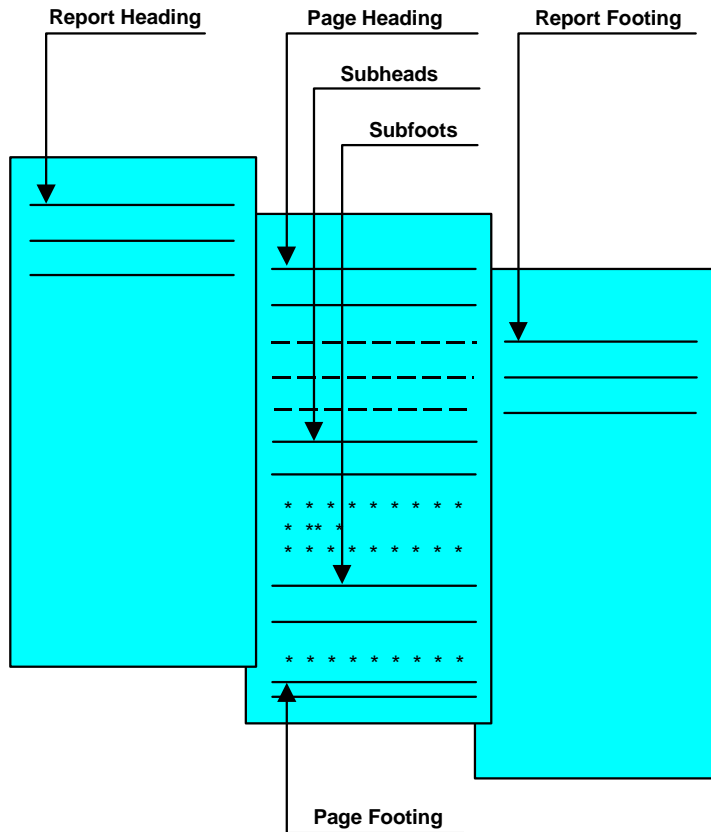
This request produces the following report:

PAGE 1

LAST_NAME	FIRST_NAME	DEPARTMENT	
		MIS	PRODUCTION
BANNING	JOHN	NONE	\$29,700.00
BLACKWOOD	ROSEMARIE	\$21,780.00	NONE
CROSS	BARBARA	\$27,062.00	NONE
GREENSPAN	MARY	\$9,000.00	NONE
IRVING	JOAN	NONE	\$26,862.00
JONES	DIANE	\$18,480.00	NONE
MCCOY	JOHN	\$18,480.00	NONE
MCKNIGHT	ROGER	NONE	\$16,100.00
ROMANS	ANTHONY	NONE	\$21,120.00
SMITH	MARY	\$13,200.00	NONE
	RICHARD	NONE	\$9,500.00
STEVENS	ALFRED	NONE	\$11,000.00

Producing Headings and Footings

You can use the variety of headings and footings to clarify the information presented in your reports. You may create up to 57 lines of headings and footings in a single report request. The following diagram illustrates the options available:



Report and Page Headings

A report heading is text that appears on the top of the first page of a report. A page heading is text that appears on the top of every page of a report. In general, the heading is composed of text that you supply in your report request, enclosed in double quotation marks.

Note: If the ending quotation mark of the heading text is omitted, all subsequent lines of the request will be treated as part of the heading.

Syntax

How to Create a Report Heading

To create a report heading, the syntax is

```
ON TABLE [PAGE-BREAK AND] SUBHEAD  
"text"
```

where:

PAGE-BREAK

Is an optional phrase that positions the report heading on a separate page, which is then followed by the first page of the report itself. If you do not use PAGE-BREAK, the report heading appears on Page 1, followed immediately by the page heading and column titles.

text

Is text that you supply between quotation marks that will appear as a heading. Each line of a heading can have 128 characters, and may occupy from 1 to 57 lines of the page (unless other footings and headings are used). The text must be on a line by itself and must immediately follow the SUBHEAD command.

Syntax

How to Create a Page Heading

To place a heading on every page of the report, the syntax is

```
TABLE FILE filename  
[HEADING [CENTER]]  
"text"
```

where:

HEADING

Is optional if you place the text before the first display command; otherwise, it is required to identify the text as a heading. The command CENTER centers the heading over the text automatically.

text

Is the text placed within quotation marks that will appear on every page. The text can be split over multiple lines and must begin on the line immediately following the HEADING command.

If you supply two or more text lines between quotation marks, the lines are automatically adjusted into pairs to provide coverage across the printed page.

To position heading text, use spot markers as described in *Positioning Text* on page 9-39.

Note: A total of 6,500 characters of heading text may be supplied in a report.

Example**Creating a Report Heading**

For example:

```
TABLE FILE EMPLOYEE
SUM GROSS
BY DEPARTMENT BY HIGHEST PAY_DATE
ON TABLE PAGE-BREAK AND SUBHEAD
"PLEASE RETURN THIS TO MARY SMITH"
END
```

This request produces the following report (only the page preceding the body of the report has the subhead):

PAGE 1

PLEASE RETURN THIS TO MARY SMITH

PAGE 2

DEPARTMENT	PAY_DATE	GROSS
-----	-----	-----
MIS	82/08/31	\$9,000.00
	82/07/30	\$7,460.00
	82/06/30	\$7,460.00
	82/05/28	\$6,649.51
	82/04/30	\$5,890.84
	82/03/31	\$3,247.75
	82/02/26	\$3,247.75
	82/01/29	\$3,247.75
	81/12/31	\$2,147.75
	81/11/30	\$2,147.75
PRODUCTION	82/08/31	\$9,523.84
	82/07/30	\$7,048.84
	82/06/30	\$7,048.84
	82/05/28	\$7,048.84
	82/04/30	\$4,959.84
	82/03/31	\$4,959.84
	82/02/26	\$4,959.84
	82/01/29	\$3,705.84
	81/12/31	\$833.33
	81/11/30	\$833.33

Example **Creating a Page Heading**

For example:

```
TABLE FILE EMPLOYEE
"ACCOUNT REPORT FOR DEPARTMENT"
PRINT CURR_SAL BY DEPARTMENT BY HIGHEST BANK_ACCT
BY EMP_ID
ON DEPARTMENT PAGE-BREAK
END
```

This request produces the following two-page report:

PAGE 1			
ACCOUNT REPORT FOR DEPARTMENT			
DEPARTMENT	BANK_ACCT	EMP_ID	CURR_SAL

MIS	163800144	818692173	\$27,062.00
	122850108	326179357	\$21,780.00
	40950036	117593129	\$18,480.00
		112847612	\$13,200.00
		219984371	\$18,480.00
		543729165	\$9,000.00
PAGE 2			
ACCOUNT REPORT FOR DEPARTMENT			
DEPARTMENT	BANK_ACCT	EMP_ID	CURR_SAL

PRODUCTION	819000702	123764317	\$26,862.00
	136500120	451123478	\$16,100.00
	160633	119329144	\$29,700.00
		071382660	\$11,000.00
		119265415	\$9,500.00
		126724188	\$21,120.00

Example

Creating a Multi-Line Heading

For example:

```
TABLE FILE PROD
"  DETAIL LISTING OF AREA SALES
  DISTRIBUTION"
"    FIRST QUARTER OF YEAR
  BRANCH MANAGERS"
BY PROD_CODE NOPRINT
END
```

The report heading across the top of each page appears as:

```
PAGE      1

DETAIL LISTING OF AREA SALES                DISTRIBUTION
FIRST QUARTER OF YEAR                      BRANCH MANAGERS
```

DISTRIBUTION and BRANCH MANAGERS are on the far right of the report because of trailing blanks in the procedure. The open and closing quote marks indicate the length of the text. To avoid extra blanks, code <0X.

Report and Page Footings

A report footing is text that appears at the bottom of the last page of a report. A page footing is text that appears on the bottom of every page of a report. In general, the footing is composed of text that you can supply, in a report request, between quotation marks.

Note: If the ending quotation mark of the footing text is omitted, all subsequent lines of the request will be treated as part of the footing.

Syntax

How to Create a Report Footing

To place a footing on the last page of the report, the syntax is

```
ON TABLE [PAGE-BREAK AND] SUBFOOT
"text"
```

where:

PAGE-BREAK

Is an optional phrase that positions the report footing on the last page by itself. If not used, the report footing appears as the last line on the report.

Note: If PAGE-BREAK is specified in the BY phrase and not in the ON TABLE phrase, the report footing appears as the last line on the last page of the report.

text

Is the text you supply in quotation marks that appears as a footing. The text begins on the line following the keyword SUBFOOT. Each line of a footing can be 128 characters in length, and may occupy from one to 57 lines of the page (unless other footings and headings are used).

Syntax

How to Create a Page Footing

To display a footing on every page of a report, the syntax is

```
FOOTING [CENTER] [BOTTOM]
"text "
```

where:

FOOTING

Is the keyword that identifies the text as a footing.

CENTER

Centers the footing automatically.

BOTTOM

Places the footing at the bottom of the page. If BOTTOM is not specified, the footing text appears two lines below the report.

text

Is the text you place within quotation marks that will appear on every page. Each line of a footing can be 128 characters in length, and may occupy from one to 57 lines of the page.

Example

Creating a Page Footing

For example, this request

```
TABLE FILE CAR
WRITE SALES BY COUNTRY
FOOTING
"THIS IS HOW A FOOTNOTE IS ADDED TO EACH"
"PRINTED PAGE"
END
```

The output is:

PAGE	1
COUNTRY	SALES
-----	-----
ENGLAND	12000
FRANCE	0
ITALY	30200
JAPAN	78030
W GERMANY	88190
THIS IS HOW A FOOTNOTE IS ADDED TO EACH	
PRINTED PAGE	

Subheads

A subhead is text that can be placed before the sort field values change.

Note: If the ending quotation mark of the subheading text is omitted, all subsequent lines of the request will be treated as part of the subheading.

Syntax

How to Create a Subhead

The syntax is

```
{ON|BY}  fieldname SUBHEAD  
"text"  
[WHEN expression;
```

where:

fieldname

Is the sort field before which the text will be inserted.

text

Is the text you supply between double quotation marks that will be printed following the SUBHEAD phrase. The total number of subheads and subfoots in one report request may not exceed nine.

WHEN *expression*

Specifies a conditional subhead in the printing of a report, as determined by a Boolean expression. Used with SUBHEAD, the WHEN clause must be placed on a line following the text you enclose in double quotation marks.

Example **Using Subheads**

For example, this request

```
TABLE FILE PROD
SUM PACKAGE AND UNIT_COST
BY PROD_NAME NOPRINT BY PROD_CODE
ON PROD_NAME SUBHEAD
" SUMMARY FOR <PROD_NAME"
END
```

creates the report below:

PAGE 1

PROD_CODE	PACKAGE	UNIT_COST
SUMMARY FOR AMERICAN CHEESE		
C7	8 OUNCES	\$2.19
SUMMARY FOR BUTTER MILK		
C14	32 OUNCES	\$1.89
SUMMARY FOR CHEDDAR CHEESE		
B19	7 OUNCES	\$.95
SUMMARY FOR CHOCOLATE MILK		
B20	32 OUNCES	\$1.79
SUMMARY FOR HEAVY CREAM		
C17	32 OUNCES	\$1.89
SUMMARY FOR LARGE EGGS		
E2	ONE DOZEN	\$.79
SUMMARY FOR MEDIUM EGGS		
E1	ONE DOZEN	\$.59
SUMMARY FOR SALTED BUTTER		
D15	8 OUNCES	\$.69
SUMMARY FOR SOUR CREAM		
C13	16 OUNCES	\$1.49
SUMMARY FOR SWISS CHEESE		
B17	16 OUNCES	\$1.65
SUMMARY FOR WHIPPED BUTTER		
D12	16 OUNCES	\$1.79
SUMMARY FOR WHOLE MILK		
B10	16 OUNCES	\$.65
B12	32 OUNCES	\$1.15
SUMMARY FOR X-LARGE EGGS		
E3	ONE DOZEN	\$.89

Subfoots

A subfoot is text that can be placed after the sort field values change. MULTILINES can also be used with SUBFOOT to suppress SUBFOOTs.

Note: If the ending quotation mark of the subfooting text is omitted, all subsequent lines of the request will be treated as part of the subfooting.

Syntax

How to Create Subfoots

The syntax is

```
{ON|BY}  fieldname SUBFOOT [MULTILINES]
" text "
[WHEN expression ;]
```

where:

fieldname

Is the field after which the text will be inserted.

text

Is the text you supply between double quotation marks that will be printed following the SUBFOOT phrase. The total number of subheads and subfoots in one report request may not exceed nine.

MULTILINES

Is used to suppress the SUBFOOT when there is only one line of output for the BY group. Note that MULTI-LINES is a synonym for MULTILINES.

FOCUS also allows you to suppress grand totals using the NOTOTAL phrase as described in Chapter 7, *Including Totals and Subtotals*.

WHEN *expression*

Specifies a conditional subfoot in the printing of a report, as determined by a Boolean expression (see *Using Direct Operators in Headings and Footings* on page 9-44). Used with SUBFOOT, WHEN must be placed on the line following the text you enclose in double quotation marks.

Reference

Usage Notes for Creating Subfoots

- When a SUBFOOT follows a RECAP, the default display of the RECAP values is suppressed, as it is assumed that the SUBFOOT is being used to display the RECAP. Notice that the phrase <DEPAR_NET was replaced with the actual value of DEPAR_NET.
- A SUBFOOT can also be used as a complete report request without any display command if data is embedded in the text, because fields in the text become implicit display fields. The default display command is SUM. For more information, see *Using Data in Headings and Footings* on page 9-41.
- If the report request contains the command SUM and the display field is specified in a subfoot, the value is summed. Use direct operators with fields specified in subfootings.

Example Using Subfoots

This example

```
TABLE FILE EMPLOYEE
SUM DED_AMT AND GROSS
BY DEPARTMENT BY HIGHEST PAY_DATE
ON DEPARTMENT RECAP
DEPAR_NET/D8.2=GROSS-DED_AMT;
ON DEPARTMENT SUBFOOT
"DEPARTMENT NET = <DEPAR_NET"
END
```

produces the following report:

PAGE 1

DEPARTMENT	PAY_DATE	DED_AMT	GROSS
MIS	82/08/31	\$4,575.76	\$9,000.00
	82/07/30	\$4,117.07	\$7,460.00
	82/06/30	\$4,117.07	\$7,460.00
	82/05/28	\$3,954.39	\$6,649.51
	82/04/30	\$3,386.76	\$5,890.84
	82/03/31	\$1,740.88	\$3,247.75
	82/02/26	\$1,740.88	\$3,247.75
	82/01/29	\$1,740.88	\$3,247.75
	81/12/31	\$1,406.78	\$2,147.75
	81/11/30	\$1,406.78	\$2,147.75
DEPARTMENT NET =	22,311.88		
PRODUCTION	82/08/31	\$4,911.14	\$9,523.84
	82/07/30	\$3,483.89	\$7,048.84
	82/06/30	\$3,483.89	\$7,048.84
	82/05/28	\$3,483.89	\$7,048.84
	82/04/30	\$2,061.70	\$4,959.84
	82/03/31	\$2,061.70	\$4,959.84
	82/02/26	\$2,061.70	\$4,959.84
	82/01/29	\$1,560.10	\$3,705.84
	81/12/31	\$141.66	\$833.33
	81/11/30	\$141.66	\$833.33
DEPARTMENT NET =	27,531.03		

Positioning Text

The positioning of text and data in headings, footings, subheads, and subfoots can be controlled by a spot marker, which identifies the column where the text should begin. A spot marker consists of a left caret (<) followed by a number indicating the absolute or relative column position. The right caret (>) is optional and can make the spot marker clearer to a reader.

The various ways spot markers can be used are illustrated in the chart below:

Marker	Example	Usage
<n or <n>	<50	The next character starts in column 50.
<+n or <+n>	<+4	The next character starts four columns from the last non-blank character.
<-n or <-n>	<-1	The next character starts one column to the left of the last character and suppresses or writes over all or part of a field.
</n or </n>	</2	Skip two lines.
<0X or <0X>	<0X	Positions the next character immediately to the right of the last character (skip zero columns). This is used when you have more than two lines between the double quotation marks in a stored procedure that make up a single line of heading, subhead, footing, or subfoot display. No spaces are inserted between the spot marker and the start of a continuation line.

Note: If you place a skip line spot marker on a line by itself, it will skip one more line than you asked for. To avoid this, put the skip line marker on the same line with additional text from the report. In addition, each field needs one space for field attributes; if a field placed with a spot marker overlaps an existing field, unpredictable results may occur.

Example

Positioning Text

- To place a character in a specific column:

```
"<50 SUMMARY REPORT"
```

The letter S in SUMMARY will start in Position 50 of the line.

- To place a substituted value in a specific column:

```
"<15 COST OF VEHICLE IS <40 <RCOST>"  
"<10 <DIVISION <30 <AREA <50 <DATE"
```

- To add spaces to the right of the last non-blank character:

```
"DAILY REPORT <DATE <+5 <LOCATION <+5 <PRODUCT"
```

- To move to the left of the last non-blank character:

```
"<60 CONFIDENTIAL <-40 <FIRST_NAME"
```

Skipping backward may cover other text on a line. This may be useful in some cases, but in general should be avoided.

- To show four lines of heading text between double quotation marks:

```
"THIS HEADING <0X  
SHOULD APPEAR <0X  
ON ONE <0X  
LINE"
```

The above produces the line:

```
THIS HEADING SHOULD APPEAR ON ONE LINE
```

- To position a long line:

```
"<20 DETAIL REPORT WITH LOTS OF TEXT ON ONE LINE  
<100 EVEN THOUGH IT IS ON TWO LINES IN THE REQUEST"
```

- To skip multiple lines:

```
"</4 THIS IS ON THE FIFTH LINE DOWN"
```


Using Data in Headings and Footings

You can embed the values of fields in headings, subheads, subfoots, and footings.

Syntax

How to Insert Data in Headings and Footings

To put a value in one of these titles, use the following syntax:

```
<fieldname  
<fieldname>
```

where:

```
<fieldname
```

Places the data value in the heading or footing, and suppresses trailing blanks.

```
<fieldname>
```

Places the data value in the heading or footing, and retains trailing blanks.

Reference

Usage Notes for Data in Headings and Footings

- Trailing blanks in alphanumeric fields may be omitted by using only the opening < character for data in headings. For example, if AREA is a 16-character alphanumeric field, the line is expanded by 16 characters at the point of substitution of the retrieved value. If only the opening character is used, only the non-blank characters of the particular value are substituted. For example, if <AREA retrieves the value of EAST, only four characters plus one leading blank are inserted in the line, rather than a full 16 characters which the data value could contain.
- A SUBFOOT can be used as a complete report request without any display command if data is embedded in the text, because fields in the text become implicit display fields.
- You can place page numbers in headings and footings using TABPAGENO (see *Inserting Page Numbers: TABPAGENO* on page 9-4).
- Fields in headings and footings are evaluated as if they were verb objects of the first verb. Fields in subheads and subfoots are evaluated as part of the first verb in which they are referenced. If a field is not referenced, it is evaluated as part of the last verb.
- Text fields (FORMAT=TXnn) can be embedded in a heading or footing:
 - Text field values may display on multiple lines. The output is aligned vertically so that the position of the field on the initial line is maintained on the following lines.
 - The number of characters in the TX format specification determines the number of spaces per line for the field in the heading or footing.
 - HEADING and FOOTING lines can contain multiple TX fields. SUBHEAD and SUBFOOT lines can contain at most one.
 - You cannot embed TX fields in FML free-text lines.

Example Using Data in a Heading

For example:

```
TABLE FILE EMPLOYEE
"<DEPARTMENT>: BANK, EMPLOYEES AND SALARIES </1"
PRINT CURR_SAL
BY DEPARTMENT NOPRINT BY BANK_ACCT
BY LAST_NAME BY FIRST_NAME
ON DEPARTMENT PAGE-BREAK
FOOTING
"<DEPARTMENT EMPLOYEES WITH ELECTRONIC TRANSFER ACCOUNTS"
END
```

This request produces the following 2-page report:

PAGE 1

MIS : BANK, EMPLOYEES AND SALARIES

BANK_ACCT	LAST_NAME	FIRST_NAME	CURR_SAL
	GREENSPAN	MARY	\$9,000.00
	MCCOY	JOHN	\$18,480.00
	SMITH	MARY	\$13,200.00
40950036	JONES	DIANE	\$18,480.00
122850108	BLACKWOOD	ROSEMARIE	\$21,780.00
163800144	CROSS	BARBARA	\$27,062.00

MIS EMPLOYEES WITH ELECTRONIC TRANSFER ACCOUNTS

PAGE 2

PRODUCTION : BANK, EMPLOYEES AND SALARIES

BANK_ACCT	LAST_NAME	FIRST_NAME	CURR_SAL
	ROMANS	ANTHONY	\$21,120.00
	SMITH	RICHARD	\$9,500.00
	STEVENS	ALFRED	\$11,000.00
160633	BANNING	JOHN	\$29,700.00
136500120	MCKNIGHT	ROGER	\$16,100.00
819000702	IRVING	JOAN	\$26,862.00

PRODUCTION EMPLOYEES WITH ELECTRONIC TRANSFER ACCOUNTS

Example**Using Data in a SUBFOOT**

For example, this request:

```
TABLE FILE CAR
BY COUNTRY NOPRINT SUBFOOT
"NUMBER OF MODELS IN COUNTRY <COUNTRY = <CNT.MODEL
WITH AVERAGE COST OF <AVE.RCOST "
END
```

creates this report:

PAGE 1

NUMBER OF MODELS IN COUNTRY ENGLAND =	4	WITH AVERAGE COST OF	11,330
NUMBER OF MODELS IN COUNTRY FRANCE =	1	WITH AVERAGE COST OF	5,610
NUMBER OF MODELS IN COUNTRY ITALY =	4	WITH AVERAGE COST OF	12,766
NUMBER OF MODELS IN COUNTRY JAPAN =	2	WITH AVERAGE COST OF	3,239
NUMBER OF MODELS IN COUNTRY W GERMANY =	7	WITH AVERAGE COST OF	9,247

Example **Using Direct Operators in Headings and Footings**

You can use any prefix operator in a heading or footing to perform specific operations. Consider the following examples:

```
TABLE FILE SALES
"MOST UNITS SOLD WERE <MAX.UNIT_SOLD"
"LEAST UNITS SOLD WERE <MIN.UNIT_SOLD"
"AVERAGE UNITS SOLD WERE <AVE.UNIT_SOLD"
"TOTAL UNITS SOLD WERE <TOT.UNIT_SOLD"
END
```

PAGE 1

MOST UNITS SOLD WERE	80
LEAST UNITS SOLD WERE	12
AVERAGE UNITS SOLD WERE	33
TOTAL UNITS SOLD WERE	645

and

```
TABLE FILE EMPLOYEE
"EMPLOYEE NAME <FIRST_NAME <LAST_NAME"
"CURRENT DEPARTMENT       <DEPARTMENT"
"JOB TITLE <JOB_DESC"
"*****"
"SKILL CATEGORY            <SKILLS"
"*****"
" "
WHERE EMP_ID IS '112847612'
END
```

PAGE 1

EMPLOYEE NAME	MARY SMITH
CURRENT DEPARTMENT	MIS
JOB TITLE	FILE QUALITY

SKILL CATEGORY	FIQU

and

```

DEFINE FILE SALES
ACTUAL_SALES/D8.2 = UNIT_SOLD-RETURNS;
%SALES/F5.1 = 100*ACTUAL_SALES/UNIT_SOLD;
END

TABLE FILE SALES
"SUMMARY OF ACTUAL SALES"
"UNITS SOLD      <TOT.UNIT_SOLD"
"RETURNS          <TOT.RETURNS"
"      ===== "
"TOTAL SOLD      <TOT.ACTUAL_SALES"
" "
"BREAKDOWN BY PRODUCT"
PRINT UNIT_SOLD AND RETURNS AND ACTUAL_SALES
BY PROD_CODE
END

```

The following shows the beginning of the output:

```

SUMMARY OF ACTUAL SALES
UNITS SOLD      645
RETURNS          58
      =====
TOTAL SOLD      587.00

BREAKDOWN BY PRODUCT
PROD_CODE  UNIT_SOLD  RETURNS  ACTUAL_SALES
-----
B10                60        10        50.00
                  30         2        28.00
                  13         1        12.00
B12                40         3        37.00

```

You can use the following special operators only in subheadings and subfootings:

ST.fieldname

Produces a subtotal value of the specified field at a sort break in the report.

CT.fieldname

Produces a cumulative total of the specified field.

Producing a Free-Form Report

Report requests do not have to produce a tabular display, but may consist of only the heading (as long as the heading has a data field referenced in it). If the request has no display command but there is a data field embedded in the heading, FOCUS assumes that this is a heading only request and does not print the body of the report. Any data fields referenced in the heading will be treated as if they were display fields. Their values at the time the heading is printed are what they would have been had they been mentioned as in a display command. Free-form reports are described in detail in Chapter 17, *Creating a Free-Form Report*.

Conditionally Formatting Reports With the WHEN Clause

Use the WHEN clause in a TABLE request to conditionally display summary lines and formatting options for BY fields. The expression in the WHEN clause enables you to control where options such as SUBTOTAL and SUBFOOT appear in the report.

The WHEN clause is an extension of the ON phrase and must follow the ON phrase to which it applies. One WHEN clause can be specified for each option in the ON phrase. Multiple WHEN clauses are also permitted.

Used with certain formatting options in a TABLE request, the WHEN clause controls when those formatting options are displayed. If a WHEN clause is not used, the formatting options are displayed whenever the sort field value changes.

Syntax

How to Create Conditional Formatting

Syntax for the WHEN clause is

```
ON fieldname option WHEN expression[;]
```

where:

option

Is any one of the following options for the ON phrase in a TABLE:

RECAP	PAGE-BREAK	SUBHEAD
RECOMPUTE	REPAGE	SUBFOOT
SUBTOTAL	SKIP-LINE	SUB-TOTAL
UNDER-LINE	SUMMARIZE	

If the WHEN clause is used with SUBHEAD or SUBFOOT, it must be placed on the line following the text that is enclosed in double quotation marks (see *Producing Headings and Footings* on page 9-29).

expression

Is any Boolean expression that would be valid on the right side of a COMPUTE expression (see Chapter 8, *Using Expressions*).

Note:

- IF ... THEN ... ELSE logic is not necessary in a WHEN clause and is not supported.
- All non-numeric literals in a WHEN expression must be specified with single quotation marks.
- The semicolon at the end of a WHEN expression is optional and may be included for readability.

Reference**Usage Notes for Conditional Formatting**

- A separate WHEN clause may be used for each option specified in an ON phrase. The ON field name phrase needs to be specified only once.
- You can use the WHEN clause to display a different SUBFOOT or SUBHEAD for each break group.
- The WHEN clause only applies to the option that immediately precedes it.
- If a WHEN clause specifies an aggregated field, the value tested is aggregated only within the break determined by the field in the corresponding ON phrase.
- In the WHEN clause for a SUBFOOT, the SUBTOTAL is calculated and evaluated. This applies to fields with prefix operators and to summed fields. For alphanumeric fields, the last value in the break group is used in the test.

Example**Conditionally Formatting Reports**

In the following example, the WHEN clause prints a subfoot at the break for the field STORE_CODE only when the sum of PRODSALES exceeds \$500:

```
DEFINE FILE SALES
PRODSALES/D9.2M = UNIT_SOLD * RETAIL_PRICE;
END

TABLE FILE SALES
SUM PRODSALES
BY STORE_CODE
ON STORE_CODE SUBFOOT
"*** SALES FOR STORE <STORE_CODE EXCEED $500 ***"
WHEN PRODSALES GT 500
END
```

The report output looks like this:

```
PAGE      1

STORE_CODE  PRODSALES
-----
K1          $56.08
14B         $535.34
*** SALES FOR STORE 14B EXCEED $500 ***
14Z         $224.88
77F         $151.85
```

Example **Selectively Displaying a SUBTOTAL and SUBFOOT**

For example, you can print a report that selectively displays a SUBTOTAL and a SUBFOOT:

```
TABLE FILE SALES
PRINT UNIT_SOLD RETAIL_PRICE
BY PROD_CODE
ON PROD_CODE SUBTOTAL
WHEN PROD_CODE CONTAINS 'B'      First WHEN phrase
SUBFOOT
"PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP"
WHEN PROD_CODE CONTAINS 'C'      Second WHEN phrase
END
```

PAGE 1

PROD_CODE	UNIT_SOLD	RETAIL_PRICE
B10	60	\$.95
	30	\$.85
	13	\$.99
*TOTAL B10	103	\$2.79
B12	40	\$1.29
	29	\$1.49
*TOTAL B12	69	\$2.78
.		
.		
.		
C13	25	\$1.99
PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP		
C17	12	\$2.09
PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP		
.		
.		
.		

Example**Selectively Displaying Multiple Subheads**

In the following example, a different subhead will be displayed depending on the value of the BY field. If the value of PROD_CODE contains the literal B, C, or E, the subhead CURRENT PRODUCT LINE will be displayed. If PROD_CODE contains the literal D, the subhead DISCONTINUED PRODUCT will be displayed.

```
TABLE FILE SALES
PRINT UNIT_SOLD RETAIL_PRICE
BY PROD_CODE

ON PROD_CODE
SUBHEAD
"CURRENT PRODUCT LINE"
WHEN PROD_CODE CONTAINS 'B' OR 'C' OR 'E'

SUBHEAD
"DISCONTINUED PRODUCT"
WHEN PROD_CODE CONTAINS 'D'
END
```

This produces the following report:

PAGE 1

PROD_CODE	UNIT_SOLD	RETAIL_PRICE
CURRENT PRODUCT LINE		
B10	60	\$.95
	30	\$.85
	13	\$.99
CURRENT PRODUCT LINE		
B12	40	\$1.29
	29	\$1.49
CURRENT PRODUCT LINE		
B17	29	\$1.89
	20	\$1.89
CURRENT PRODUCT LINE		
B20	15	\$1.99
	25	\$2.09
CURRENT PRODUCT LINE		
C13	25	\$1.99
CURRENT PRODUCT LINE		
C17	12	\$2.09
CURRENT PRODUCT LINE		
C7	45	\$2.39
	40	\$2.49
DISCONTINUED PRODUCT		
D12	27	\$2.19
	20	\$2.09
CURRENT PRODUCT LINE		
E1	30	\$.89
CURRENT PRODUCT LINE		
E2	80	\$.99
CURRENT PRODUCT LINE		
E3	70	\$1.09
	35	\$1.09

Example **Selectively Displaying a Subfoot**

In the following example:

```
ON PROD_CODE SUBTOTAL AND SUBFOOT
"PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP"
WHEN PROD_CODE CONTAINS 'B'
```

a subtotal will be calculated for each PROD_CODE, but the subfoot will be displayed only when PROD_CODE contains the literal B.

Example **Using Aggregation in the WHEN Clause**

For example:

```
TABLE FILE SALES
SUM UNIT_SOLD
  BY STORE_CODE BY PROD_CODE

ON STORE_CODE SUBFOOT
"SELLING ABOVE QUOTA <ST.UNIT_SOLD "
" "
WHEN UNIT_SOLD GT 100
SUBFOOT
"SELLING AT QUOTA <ST.UNIT_SOLD"
" "
WHEN UNIT_SOLD GE 40 AND UNIT_SOLD LT 100
SUBFOOT
"SELLING BELOW QUOTA <ST.UNIT_SOLD"
" "
WHEN UNIT_SOLD LT 40
END
```

This request produces the following report:

PAGE 1

STORE_CODE	PROD_CODE	UNIT_SOLD
-----	-----	-----
K1	B10	13
	B12	29
SELLING AT QUOTA	42	
14B	B10	60
	B12	40
	B17	29
	C13	25
	C7	45
	D12	27
	E2	80
	E3	70
SELLING ABOVE QUOTA	376	
14Z	B10	30
	B17	20
	B20	15
	C17	12
	D12	20
	E1	30
	E3	35
SELLING ABOVE QUOTA	162	
77F	B20	25
	C7	40
SELLING AT QUOTA	65	

Controlling the Display of Empty Reports

The SET command, SET EMPTYREPORT, enables you to control the output generated when a TABLE request retrieves zero records.

Syntax

How to Control Empty Reports

Use this command:

```
SET EMPTYREPORT = {ON|OFF}
```

Valid values are:

[ON](#)

Generates an empty report when zero records are found.

[OFF](#)

Does not generate a report when zero records are found. OFF is the default setting.

The command may also be issued from a request. For example:

```
ON TABLE SET EMPTYREPORT ON
```

Reference

Usage Notes for Displaying Empty Reports

- TABLEF is not supported with SET EMPTYREPORT. When a TABLEF request retrieves zero records, EMPTYREPORT behaves as if EMPTYREPORT is set ON.
- This is a change in default behavior from prior releases of FOCUS. To restore prior default behavior, issue the SET EMPTYREPORT = ON command.
- SET EMPTYREPORT = OFF is not supported for HOLD FORMAT WP files.
- SET EMPTYREPORT = ON behaves as described regardless of ONLINE or OFFLINE settings.

CHAPTER 10

Styling Reports: StyleSheets

Topics:

- Introduction to StyleSheets
- What Is a StyleSheet?
- Creating a StyleSheet
- Printing Styled Reports
- Styling the Page Layout
- StyleSheet Files
- Identifying Report Components
- StyleSheet Inheritance
- Conditional Styling

With StyleSheets, you can produce visually interesting, presentation-quality reports that highlight key information. You can choose the fonts, colors, and text styles for individual report components, either unconditionally or based on data values and conditions. You can also move columns anywhere in the report and control page parameters such as margins, size, and orientation.

For example, you can make a column title 18 point Times, make data values in the same column 16 point Helvetica with different colors for values that satisfy certain conditions, and you can italicize the column total. You can also move that column anywhere in the report.

Introduction to StyleSheets

StyleSheets provide numerous options, so you can create extremely detailed formats for every line, column, or value in your report. This powerful customization and its detailed syntax makes the process of using StyleSheets seem more complicated than it actually is. In most cases, you will want to use the formatting facilities judiciously to make important information stand out. The process of using a StyleSheet consists of the following steps:

1. Decide which StyleSheet to use. If one already exists with the formatting you need, go on to the next step. FOCUS comes with default styles that you can use if you want the whole report printed with the same default format. If you want to customize certain formats in your report, create a StyleSheet that describes those formats.

Note: If the only attributes you want to change are page layout parameters, you can change the page parameters with a SET command or in a report request. *Styling the Page Layout* on page 10-13 discusses page layout parameters.

2. Activate the StyleSheet you have chosen or create a StyleSheet in a report request.
3. Create a PostScript file that contains the formatted report.
4. Print the report on a PostScript printer. This step is highly dependent on the equipment and software at your site. See your system administrator for instructions.

The following sample report shows how StyleSheets can enhance the usefulness and appearance of your reports:

PAGE 1

**Swifty Information Group
Department Spending Report**

By Department, By Pay Date

DEPARTMENT	Pay Date	Bank Account	Gross Amount	Deduction Amount	Deduction Gross Ratio
MIS	82/08/31	40950036	\$1,540.00	\$725.35	.47
		122850108	\$1,815.00	\$1,261.42	.69
		163800144	\$2,255.00	\$1,668.70	.74
	82/07/30	40950036	\$1,540.00	\$725.35	.47
		122850108	\$1,815.00	\$1,261.42	.69
		163800144	\$2,255.00	\$1,668.70	.74
	82/06/30	40950036	\$1,540.00	\$725.35	.47
		122850108	\$1,815.00	\$1,261.42	.69
		163800144	\$2,255.00	\$1,668.70	.74
	82/05/28	40950036	\$1,479.50	\$690.17	.47
		122850108	\$1,815.00	\$1,261.42	.69
		163800144	\$2,255.00	\$1,668.70	.74
	82/04/30	122850108	\$1,815.00	\$1,261.42	.69
		163800144	\$2,255.00	\$1,668.70	.74
		163800144	\$2,147.75	\$1,406.78	.66
	82/02/26	163800144	\$2,147.75	\$1,406.78	.66
*TOTAL DEPARTMENT MIS			\$30,745.01	\$20,330.40	.66
MIS Department Net Earnings:			\$10,414.61		
PRODUCTION	82/08/31	160633	\$2,475.00	\$1,427.25	.58
		136500120	\$1,342.00	\$522.28	.39
		819000702	\$2,238.50	\$1,746.04	.78
	82/07/30	136500120	\$1,342.00	\$522.28	.39
		819000702	\$2,238.50	\$1,746.04	.78
		136500120	\$1,342.00	\$522.28	.39
	82/06/30	819000702	\$2,238.50	\$1,746.04	.78
		136500120	\$1,342.00	\$522.28	.39
		819000702	\$2,238.50	\$1,746.04	.78
	82/05/28	136500120	\$1,342.00	\$522.28	.39
		819000702	\$2,238.50	\$1,746.04	.78
		136500120	\$1,254.00	\$501.60	.40
	82/04/30	819000702	\$2,035.00	\$1,241.35	.61
		136500120	\$1,254.00	\$501.60	.40
		819000702	\$2,035.00	\$1,241.35	.61
	82/03/31	136500120	\$1,254.00	\$501.60	.40
		819000702	\$2,035.00	\$1,241.35	.61
		136500120	\$1,254.00	\$501.60	.40
	82/02/26	819000702	\$2,035.00	\$1,241.35	.61
*TOTAL DEPARTMENT PRODUCTION			\$26,663.99	\$15,729.37	.59
PRODUCTION Department Net Earnings:			\$10,934.62		
TOTAL			\$57,409.00	\$36,059.78	.63

Confidential Information

This section describes:

- Basic StyleSheet concepts, and requirements for using StyleSheets (see *What Is a StyleSheet?* on page 10-5).
- How to use an existing StyleSheet, either a custom StyleSheet or the default StyleSheet that is included with FOCUS (see *Activating an Existing StyleSheet File* on page 10-11).
- How to print styled reports (see *Printing Styled Reports* on page 10-12).
- Page layout settings (see *Styling the Page Layout* on page 10-13).
- Report components (see *Identifying Report Components* on page 10-21).
- StyleSheet file syntax and instructions for checking a StyleSheet (see *StyleSheet File* on page 10-17).
- Style definition syntax (see *Style Definition* on page 10-19).
- Instructions for selecting a report component, controlling column placement, and determining inheritance (see *Selecting and Manipulating Report Components* on page 10-24).
- Conditional styling with WHEN (see *Conditional Styling* on page 10-49).

What Is a StyleSheet?

A StyleSheet is a text file that describes how you want your report to look. It consists of a series of declarations that:

- Identify a report component.
- Describe the desired characteristics of that component.

You can create a StyleSheet using any text editor, including TED, the FOCUS text editor.

If you do not create a StyleSheet, FOCUS uses the same default style for all report components.

If you create a StyleSheet, you only have to define the styles of those components that you want printed differently from the default style. Any component that you do not specifically format in your StyleSheet either uses the default style or inherits a style from a higher-level component. Inheritance is discussed in *StyleSheet Inheritance* on page 10-43.

When you use a StyleSheet, your page layout differs from a FOCUS report that does not use a StyleSheet. This variation is a function of the page layout parameters FOCUS uses for the two kinds of reports:

- For reports without StyleSheets, FOCUS uses the parameters LINES, PAPER, PANEL, and WIDTH to define the page layout.
- For reports with StyleSheets, FOCUS uses the parameters PAGESIZE, ORIENTATION, UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, and SQUEEZE. Any of these parameters that you do not specifically set—either in a report request, in a StyleSheet, or via a SET command—inherits default values.

What Is a Style?

A style is a description of the physical characteristics of a report component; it consists of four attributes:

- A font (typeface) such as Helvetica, Courier, or Times.
- A size for the font such as 12-point or 14-point.
- A text style or combination of text styles such as bold, italic, or bold+italic.
- A color.

When You Need to Create a StyleSheet File

You can take advantage of most StyleSheet options without ever having to create a StyleSheet file.

You can select a StyleSheet, page size, orientation, and margins at the FOCUS command level (if you want to apply them to your entire FOCUS session), or in a report request (if you want to apply them to one report).

You need to create a StyleSheet file if you wish to:

- Style report components individually.
- Apply different styles to the same component based on report values.

Reference

Requirements for Printing Styled Reports

To print reports using a StyleSheet file, you need:

- A PostScript printer.
- Adobe Font Metrics (AFM) files that define the measurements of characters for PostScript output.

AFM files are distributed with FOCUS.

To use a StyleSheet file, FOCUS needs a font location file that maps the font names to the Font Metric files. If this file is unavailable and you request a StyleSheet, the following error results:

```
ERROR AT OR NEAR LINE      19 IN PROCEDURE focexec name FOCEXEC *  
(FOC3222) THE FONT LOCATION FILE IS MISSING:
```

Reports that use StyleSheets can reference a maximum of 128 verb objects. If the report exceeds 128 verb objects, the following error is produced:

```
(FOC3228) STYLED REPORTS CURRENTLY LIMITED TO 128 COLUMNS.
```

Reference

Reproducing StyleSheet Examples

The examples presented in this chapter use data from the EMPLOYEE and CAR databases. In order to reproduce the examples, these sample databases must be loaded on your machine.

The sample reports use the default page size specification, LETTER, which represents 8.5 x 11 inch pages. They have been scaled down to accommodate the size of this manual.

If the fonts you have on your system do not include the ones used in the examples, substitute suitable and available fonts before you run the examples.

Reference

Required Files and DDNAMES

When you produce a report using a StyleSheet, you need the following files:

Name	Purpose	MVS	CMS
StyleSheet files. You can create them with a text editor (see <i>StyleSheet File</i> on page 10-17).	Define the styles in reports.	Any member of the PDS allocated to ddname FOCSTYLE.	File type FOCSTYLE, any file name
Adobe Font Metrics (AFM) files (supplied with FOCUS).	Define the measurements of characters for PostScript output.	Member names start with PS. Allocated to ddname ERRORS.	File type AFM, any file name
Font location file (supplied with FOCUS).	Maps font names to the Font Metrics files.	Member PSCRIPT in the PDS allocated to ddname ERRORS.	File name PSCRIPT, File type FOCFTMAP
PostScript output files. You create these with a HOLD, SAVE, or SET command (see <i>Printing Styled Reports</i> on page 10-12).	Contain the formatted output for the PostScript printer.	Any member of the variable length PDS allocated to ddname PS.	File type PS, any file name

Comparison of Reports With and Without StyleSheets

In a non-styled FOCUS report request, you can set values for the maximum number of lines on the output page (LINES), the number of lines on the printed page (PAPER), the maximum number of characters in a report panel (PANEL), and the maximum number of characters on an output line (WIDTH). FOCUS then uses the typeface and size defined by your printer setup for all data in the report.

In a styled report, you can set margins on the top, bottom and sides of the report, you can set the page size for letters, envelopes, and many other types of paper, you can specify measurement units such as inches or points, and you can control column widths or spacing. You can also change typefaces, type size, and type style for any part of the report.

The following table compares what you can do with and without StyleSheets:

	With StyleSheets	Without StyleSheets
Text font	You can use different font sizes and fonts. You can selectively apply text styles such as bold or italics.	FOCUS uses the default font specified in your printer setup for the entire report.
Colors	You can select a color for the text.	The ink and paper in your printer determine the colors in your report.
Individual components	You can assign different styles to individual report components.	FOCUS uses a single style for the entire report.
Conditional styling	You can apply different styles to the same component based on report values.	Report format does not change with changes in report values.
Column widths	You can have column widths based on the column content, the field format specified in the Master File, or specify a width.	FOCUS bases column widths on the column title or the field format specified either in the Master File or the report request.
Column placement	You can specify the starting position of individual columns and arrange columns in any order regardless of the sequence established in the report request. In addition, you can indicate how much space to leave before and after individual columns.	You can specify the starting position of individual columns in the report request.
Page size	You can select from a wide range of page sizes.	You can specify the number of lines per page and characters per line within the limits of your printer setup.
Page orientation	You can select either portrait or landscape.	FOCUS uses the default orientation specified in your printer setup.
Page margins	You can specify the top, bottom, left, and right margins, measured in inches, centimeters, or points.	FOCUS uses the default page margins specified in your printer setup.

Creating a StyleSheet

There are two ways to create a StyleSheet:

- You can create a StyleSheet file in a report request. This is useful when you are applying that set of styles to only one report. See *Creating a StyleSheet Within a Report Request* on page 10-9.
- You can create a separate StyleSheet file. In order to use a StyleSheet file, you must activate it. This option is useful when you want to create a StyleSheet template that you can apply to any report. In addition, you can create a StyleSheet on one platform and then port it to and run it on other platforms. See *Activating an Existing StyleSheet File* on page 10-11.

Creating a StyleSheet Within a Report Request

You can create a StyleSheet file within your report request. This method enables you to create and maintain the styles for your report directly in the report request.

Syntax

How to Create a StyleSheet in a Report Request

```
ON TABLE SET STYLE *  
.  
.  
.  
ENDSTYLE
```

where:

`STYLE *`

Indicates the beginning of an inline StyleSheet.

`ENDSTYLE`

Indicates the end of an inline StyleSheet.

Note: You can omit the keyword `ENDSTYLE`, but only if it is immediately followed by the keyword `END` in the report request.

Example **Creating a StyleSheet Within a Report Request**

In the following report request, the StyleSheet syntax appears in bold.

```
TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME
BY DEPARTMENT
ON TABLE HOLD FORMAT PS
ON TABLE SET STYLE *
TYPE=REPORT, FONT=TIMES, SIZE=10, $
TYPE=REPORT, COLUMN=EMP_ID, RIGHTGAP=1, $
ENDSTYLE
END
```

The request produces the following report:

PAGE 1

<u>DEPARTMENT</u>	<u>EMP_ID</u>	<u>FIRST_NAME</u>	<u>LAST_NAME</u>
MIS	112847612	MARY	SMITH
	117593129	DIANE	JONES
	219984371	JOHN	MCCOY
	326179357	ROSEMARIE	BLACKWOOD
	543729165	MARY	GREENSPAN
PRODUCTION	818692173	BARBARA	CROSS
	071382660	ALFRED	STEVENS
	119265415	RICHARD	SMITH
	119329144	JOHN	BANNING
	123764317	JOAN	IRVING
	126724188	ANTHONY	ROMANS
	451123478	ROGER	MCKNIGHT

Activating an Existing StyleSheet File

The `STYLESHEET` parameter setting determines whether a report uses a custom StyleSheet. It also determines which page layout parameters are activated.

You can specify the `STYLESHEET` setting with a `SET` command if you want the setting to apply to your entire `FOCUS` session, or in a report request if you want the setting to apply to just that report.

Syntax

How to Activate an Existing StyleSheet File

The syntax for the `SET` command is

```
SET STYLE[SHEET] = styoption
```

and the syntax in a report request is

```
TABLE FILE file
    request
ON TABLE SET STYLE styoption
END
```

where:

styoption

Is one of the following options:

ON uses default styles. This is the default setting. With this setting in effect, `FOCUS` uses the page layout settings for `UNITS`, `TOPMARGIN`, `BOTTOMMARGIN`, `LEFTMARGIN`, `RIGHTMARGIN`, `PAGESIZE`, `ORIENTATION`, and `SQUEEZE`, and `FOCUS` ignores the settings for `LINES`, `PAPER`, `PANEL`, and `WIDTH`.

OFF uses default styles. In this case, `FOCUS` uses the settings for `LINES`, `PAPER`, `PANEL`, and `WIDTH`, and it ignores the settings for `UNITS`, `BOTTOMMARGIN`, `LEFTMARGIN`, `RIGHTMARGIN`, `TOPMARGIN`, `PAGESIZE`, `ORIENTATION`, and `SQUEEZE`. The report is printed in fixed-width Courier typeface with .250-inch margins. You can use this setting to print traditional-looking `FOCUS` reports on PostScript printers.

Note: To disable StyleSheets entirely so that no StyleSheet is activated, use the `ONLINE-FMT` setting discussed in *Printing Styled Reports* on page 10-12.

stysheet is the eight-character name of a StyleSheet file. This setting activates the named StyleSheet. `FOCUS` uses the page layout settings for `UNITS`, `TOPMARGIN`, `BOTTOMMARGIN`, `LEFTMARGIN`, `RIGHTMARGIN`, `PAGESIZE`, `ORIENTATION`, and `SQUEEZE`, and `FOCUS` ignores the settings for `LINES`, `PAPER`, `PANEL`, and `WIDTH`.

Page layout settings are discussed in *Styling the Page Layout* on page 10-13, and custom StyleSheets are described in *StyleSheet File* on page 10-17.

Printing Styled Reports

In order to print a report that was formatted using a StyleSheet, you must generate a PostScript file containing the formatted report. You must then print this file on a PostScript printer. When you view the report on your screen, you do not see the special formatting unless you have a PostScript viewer that can produce the formats on-line.

Syntax

How to Print Styled Reports

You can generate StyleSheet report output in either of two ways:

- Create a HOLD or SAVE file containing the report output in PostScript format. You can issue the HOLD or SAVE command either from the FOCUS command line or in a TABLE request. The syntax is

```
[ON TABLE] {HOLD|SAVE} FORMAT {PS|POSTSCRIPT} [AS filename]
```

where:

filename

Assigns a 1- to 8-character file name or ddname to the report saved in PostScript format. The default file name is HOLD. If a PDS is allocated to ddname PS, the output file becomes member *filename* in that PDS; otherwise, the output goes to a temporary sequential data set allocated to ddname *filename*.

POSTSCRIPT|PS

Stores the report output in a PostScript file.

- Issue the following command to select a printer driver

```
SET ONLINE-FMT = {STANDARD|POSTSCRIPT}
```

where:

STANDARD

Is the default. It overrides the STYLESHEET setting and simulates standard FOCUS formatting.

POSTSCRIPT

Stores the report output in a PostScript file named PSOUT. It respects the STYLESHEET setting. PS is a synonym for POSTSCRIPT.

When you generate stylized report output that is too wide to fit in the defined print area of a single page, StyleSheet formatting divides the output across multiple pages or panels. The pages are automatically numbered with decimal notation indicating the panel number (1.1, 1.2, and so on).

Prior to generating the stylized output, FOCUS displays a FOC3218 information message indicating the total width of the report and that the resulting report output spans multiple pages.

Styling the Page Layout

When a StyleSheet is activated, FOCUS uses page parameters to format the page layout. These parameters have default values that remain in effect unless you change them in one of three ways:

- With a SET command that you issue at the FOCUS command line. Your settings remain in effect for the entire FOCUS session, unless you change them. The syntax is:

```
SET parameter=value [ ,...,parameter=value ]
```

- With an ON TABLE phrase in a report request. These settings are in effect for the duration of the request and override values specified at the command line. The syntax is:


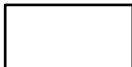
```
TABLE FILE file
    request
ON TABLE SET parameter value [ ,..., parameter value ]
END
```

- In a StyleSheet. These settings are in effect whenever you activate the StyleSheet. The values specified in an activated StyleSheet override values specified at the command line or in a report request. The syntax is:

```
[TYPE=REPORT] [parameter=value ,..., parameter=value]
```

StyleSheet file declarations are discussed more fully in *StyleSheet File* on page 10-17.

The parameters and values for styling a page layout are listed in the following chart:

Parameter Name (Default Value)	Description	Possible Values
ORIENTATION (PORTRAIT)	Specifies whether the report pages are long or wide.	<div> PORTRAIT  </div> <div> LANDSCAPE  </div>
UNITS (INCHES)	Specifies the measurement unit for margins, gaps, and column widths.	<p>INCHES</p> <p>CM (centimeters)</p> <p>PTS (points; one inch = 72 points; one cm = 28.35 points.)</p> <p>Note: If you change the UNITS setting, FOCUS preserves the margins by converting the numbers to the new measurement unit. For example, if a margin is 1 inch and you change the UNITS to CM, FOCUS converts the margin to 2.54 cm (the equivalent of 1 inch).</p>
TOPMARGIN (0.25 inch)	Sets the top boundary of report contents on a page.	The desired margin value in the measurement unit specified by the UNITS parameter. Margins are measured from the edge of the physical page defined by the PAGESIZE parameter. They must account for the printable area of a page.
BOTTOMMARGIN (0.25 inch)	Sets the bottom boundary of report contents on a page.	The desired margin value in the measurement unit specified by the UNITS parameter. Margins are measured from the edge of the physical page defined by the PAGESIZE parameter. They must account for the printable area of a page.

Parameter Name (Default Value)	Description	Possible Values
LEFTMARGIN (0.25 inch)	Sets the left boundary of report contents on a page.	The desired margin value in the measurement unit specified by the UNITS parameter. Margins are measured from the edge of the physical page defined by the PAGESIZE parameter. They must account for the printable area of a page.
RIGHTMARGIN (0.25 inch)	Sets the right boundary of report contents on a page.	The desired margin value in the measurement unit specified by the UNITS parameter. Margins are measured from the edge of the physical page defined by the PAGESIZE parameter. They must account for the printable area of a page.
SQUEEZE (ON)	Determines how column widths are assigned.	ON Column widths are the larger of the widest data value or the column title. OFF Column widths are the larger of the column title or the field format in the Master File. n Is the actual width in UNITS. This setting is available only within a StyleSheet. If the width cannot accommodate the value, FOCUS displays the part that fits and indicates that the value is truncated with an exclamation point (!) for alphanumeric data or an asterisk (*) for numeric data.

Parameter Name (Default Value)	Description	Possible Values	
PAGESIZE (LETTER)	Specifies the page size. Note: If the actual paper size does not match the PAGESIZE setting, your report will either be cropped or contain extra blank space.	Value	Dimensions
		LETTER	8.5 x 11 in
		TABLOID	11 x 17 in
		LEDGER	17 x 11 in
		LEGAL	8.5 x 14 in
		STATEMENT	5.5 x 8.5 in
		EXECUTIVE	7.5 x 10.5 in
		A3	297 x 420 mm
		A4	210 x 297 mm
		A5	148 x 210 mm
		B4	250 x 354 mm
		B5	182 x 257 mm
		FOLIO	8.5 x 13 in
		QUARTO	215 x 275 mm
		10X14	10 x 14 in
		ENVELOPE-9	3.875 x 8.875 in
		ENVELOPE-10	4.125 x 9.5 in
		ENVELOPE-11	4.5 x 10.375 in
		ENVELOPE-12	4.5 x 11 in
		ENVELOPE-14	5 x 11.5 in
		C	17 x 22 in
		D	22 x 34 in
		E	34 x 44 in
		ENVELOPE-DL	110 x 220 mm
		ENVELOPE-C3	324 x 458 mm
		ENVELOPE-C4	229 x 324 mm
		ENVELOPE-C5	162 x 229 mm
		ENVELOPE-C6	114 x 162 mm
		ENVELOPE-C65	114 x 229 mm
		ENVELOPE-B4	250 x 353 mm
		ENVELOPE-B5	176 x 250 mm
		ENVELOPE-B6	176 x 125 mm
		ENVELOPE-ITALY	110 x 230 mm
		ENVELOPE-MONARCH	3.875 x 7.5 in
		ENVELOPE-PERSONAL	3.625 x 6.5 in
		US-STANDARD-FANFOLD	14.875 x 11 in
		GERMAN-STANDARD-FANFOLD	8.5 x 12 in
		GERMAN-LEGAL-FANFOLD	8.5 x 13 in

Displaying Current Settings: The ? STYLE Query

Use the ? STYLE query to display the current settings for the STYLE SHEET parameter and all page parameters.

Syntax

How to Display Current Settings

The syntax is:

```
? STYLE
```

For example:

```
>
? style
  ONLINE-FMT STANDARD
  OFFLINE-FMT   STANDARD

  STYLE SHEET:   ON

  SQUEEZE       OFF
  PAGE SIZE     LETTER
  ORIENTATION   PORTRAIT
  UNITS         INCHES
  LEFT MARGIN   .250
  RIGHT MARGIN  .250
  TOP MARGIN    .250
  BOTTOM MARGIN .250
```

Note: OFFLINE-FMT is reserved for future use.

StyleSheet Files

StyleSheet files consists of a series of declarations that describe how you want your report to look. Each declaration:

- Identifies a report component or subcomponent.
- Describes the formatting to apply to that component.
- Optionally, if the component is a heading, footing, or column, specifies a position on the page for the component.
- Optionally, specifies the distance between columns, column sequence, and column width.
- Optionally, specifies a condition that must be true in order to apply the style. This technique is called stoplighting or conditional styling.

In your StyleSheet files, include declarations for only those components whose format you want to change. Within each declaration, include only those formatting attributes that you want to change.

StyleSheet Syntax

Each declaration in a StyleSheet file consists of attribute=value pairs separated by commas and terminated with a comma and dollar sign (,\$). The attributes that select a component or subcomponent must come first in each declaration. You can specify all other attributes in any order. The syntax is:

```
TYPE=value1, attribute2=value2, ... ,,$
```

Note:

- You can use uppercase, lowercase, or mixed-case in the StyleSheet file.
- Page layout parameters automatically apply to the whole report. Therefore, declarations that set page parameters do not require a TYPE attribute. For example, the following declarations are equivalent:

```
TYPE=REPORT, ORIENTATION=LANDSCAPE ,,$
```

```
ORIENTATION=LANDSCAPE ,,$
```

See *Styling the Page Layout* on page 10-13 for a complete description of page parameters.

- Each declaration must begin on a new line.
- A declaration can use more than one line. The terminating dollar sign indicates where the declaration ends.
- You can describe a single report element in more than one declaration.
- You can include blank spaces between the attributes, values, equal signs (=), commas, and dollar sign. You can also include blank lines. FOCUS can interpret declarations with or without blank spaces or lines.
- You can include comments, either on a declaration line after the terminating dollar sign, or on a separate comment line that begins with a dollar sign.

The attributes in the StyleSheet file identify report components, manipulate them, and define styles for formatting them. *Style Definition* on page 10-19 discusses the style definition, *Selecting and Manipulating Report Components* on page 10-24 describes how to select and manipulate report components, and *Conditional Styling* on page 10-49 describes conditional styling.

Checking StyleSheet Syntax

You can check the syntax of a StyleSheet from the FOCUS prompt with the CHECK STYLE command.

Syntax

How to Check StyleSheet Syntax

```
CHECK STYLE filename
```

where:

filename

Is the name of the StyleSheet file.

FOCUS reports any syntax errors in the StyleSheet file. It does not check whether the specified fonts are available or whether the font names are spelled correctly.

Style Definitions

Style definitions consists of four attributes, separated by commas, that you can specify in any order. Any attribute that you omit from the definition retains its current setting. The four attributes are

```
FONT= {font|COURIER} ,SIZE= {n|12} ,COLOR= {color|BLACK} ,  
STYLE= {[±]style [± style] |NORMAL}
```

where:

font

Is the typeface to apply. A typeface is a set of letters, numbers, and punctuation marks of a given design. COURIER is the default. Other fonts available in the font metrics files supplied with FOCUS are HELVETICA, Avant Garde Gothic, BOOKMAN, HELVETICA NARROW, NEW CENTURY SCHOOLBOOK, PALATINO, and TIMES.

You can apply fonts to all report components except underlines and skipped lines.

Embedded blanks are allowed. You can use single quotation marks for text that contains commas or where case is significant.

n

Is a positive integer. Text sizes are measured in points; the smaller the number, the smaller the type. The default size is 12 points.

color

Is one of the following: BLACK, WHITE, GREEN, RED, BLUE, MAROON, OLIVE, PURPLE, NAVY, YELLOW, TEAL, GRAY, SILVER, LIME, FUCHSIA, AQUA. The default is BLACK.

You can apply colors to all report components, except skipped lines. When colors are not available, most printers use shades of gray.

style

Is one of the following text styles:

NORMAL sets text styles to the original typeface design for the font being used. This is the default.

BOLD intensifies characters, makes them **darker**.

ITALIC slants characters and may make them more *script-like*.

OUTLINE prints only the outline of characters.

UNDERLINE prints a line under characters.

+

Combines two or more text styles, or adds a characteristic to the existing style. For example, the following adds italics to the existing style:

STYLE = +ITALIC

The following specifies bold and italics together:

STYLE = BOLD+ITALIC

-

Removes a characteristic from the existing style. For example, the following removes italics from the inherited style:

STYLE = -ITALIC

Identifying Report Components

The basic concept behind StyleSheets is that a report consists of several components, each of which has a specific name. A StyleSheet file consists of style declarations for those components whose styles you want to change, along with the formatting that you want to apply to those components. Any component that you do not specifically format in your StyleSheet either retains the default style or inherits a style from a higher-level component. Inheritance is discussed in *StyleSheet Inheritance* on page 10-43.

In a StyleSheet, you identify a report component with the TYPE attribute. The following chart lists all report components:

TYPE	Report Component
REPORT	The entire report.
PAGENUM	Default page numbers. Note: Styles created for page number lines do not apply to page numbers created by the TABPAGENO variable in TABLE requests. You can format TABPAGENO page numbers by defining a style for the heading or footing that contains it.
TABHEADING	A heading on the first page of a report, generated by ON TABLE SUBHEAD.
TABFOOTING	A footing on or after the last page of a report, generated by ON TABLE SUBFOOT.
HEADING	Headings at the top of each report page.
FOOTING	Footings at the bottom of each report page.
SUBHEAD	Headings before a particular sort field, generated by ON sortfield SUBHEAD.
SUBFOOT	Footings after a particular sort field, generated by ON sortfield SUBFOOT.
DATA	Report data.
TITLE	Column titles.
ACROSSTITLE	ACROSS field names (that is, field names used in ACROSS phrases).
ACROSSVALUE	ACROSS field values (that is, values of the ACROSS field). These values become column titles in the report.

TYPE	Report Component
SUBTOTAL	Totals generated by SUBTOTAL, SUB-TOTAL, RECOMPUTE, and SUMMARIZE.
GRANDTOTAL	The last total on a report, which can either be a column total generated by COLUMN-TOTAL or a grand total generated by SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE.
RECAP	Lines generated by ON field name RECAP or ON field name COMPUTE.
UNDERLINE	Underlines generated by ON field name UNDER-LINE.
SKIPLINE	Skipped lines generated by ON field name SKIP-LINE.

Selecting and Manipulating Report Components on page 10-24 contains annotated report output that illustrates most components.

Within certain components, you can select specific subcomponents. For example, within a heading, you can isolate a particular line or a particular field. You identify subcomponents with selection attributes (also called qualifiers). The following chart lists the attributes you can use to select a subcomponent:

Attribute	Use with TYPE:	Selects:
COLUMN	REPORT TITLE ACROSSVALUE DATA SUBTOTAL GRANDTOTAL	One or more columns. You can identify a column by its type and/or position within the report or by its field name.
ACROSSCOLUMN	REPORT TITLE DATA SUBTOTAL GRANDTOTAL	One or more columns within ACROSS groups. You can identify an ACROSSCOLUMN by its position in its ACROSS groups or by its field name.
ACROSS	ACROSSTITLE ACROSSVALUE	An ACROSS field. You can identify an ACROSS field by its position in the set of ACROSS phrases in the report request or by its field name.
BY	SUBTOTAL RECAP SUBHEAD SUBFOOT	A subtotal line for the selected BY field. You can identify a BY field by its position in the report or by its field name.

Attribute	Use with TYPE:	Selects:
LINE	TABHEADING TABFOOTING HEADING FOOTING SUBHEAD SUBFOOT	A line in any multi-line heading or footing. Blank and skipped lines count. You identify a line by its line number within its particular heading, footing, subheading, or subfooting.
OBJECT	TABHEADING TABFOOTING HEADING FOOTING SUBHEAD SUBFOOT	Either text phrases (OBJECT=TEXT) or fields (OBJECT=FIELD) in any heading or footing. You can qualify this value with a LINE and/or ITEM attribute.
ITEM	TABHEADING TABFOOTING HEADING FOOTING SUBHEAD SUBFOOT	A particular text phrase or field in a heading or footing. You identify a phrase or field by its position on its line in the heading or footing. You can use the LINE and/or OBJECT attributes to select a specific text phrase or field on a specific line.

For example, to choose the third column for the entire report, use the parameters:

- TYPE=REPORT
- COLUMN=3

Complete syntax definitions and examples are included in *Selecting and Manipulating Report Components* on page 10-24.

Selecting and Manipulating Report Components

This section describes how you select a specific component or subcomponent. It also explains how to position and manipulate columns and how to determine inheritance based on the StyleSheet hierarchy.

Syntax How to Select the Entire Report

To select the entire report, use the syntax:

```
TYPE = REPORT
```

Syntax How to Select Page Numbers

To select the default page numbers FOCUS supplies, use the syntax:

```
TYPE = PAGENUM
```

Note: Page numbers created by the TABPAGENO variable in a TABLE request are a subcomponent of the heading or footing that contains them. You can select these page numbers by selecting the heading or footing (see *Selecting Headings and Footings* on page 10-28).

Syntax How to Select Underlines

The only style attribute you can vary for an underline is the color. To select underlines for formatting, use the syntax:

```
TYPE = UNDERLINE
```

Syntax How to Select Skipped Lines

You can change the size (in points) of skipped lines. To select them for formatting, use the syntax:

```
TYPE = SKIPLINE
```

Syntax How to Select Report Data

To select the data values printed in the report, the syntax is:

```
TYPE = DATA
```

You can also select data in specific columns. See *Selecting Report Columns* on page 10-34 for details.

Syntax

How to Select Column Titles

StyleSheet declarations can distinguish between three types of column titles:

TYPE	Definition
TITLE	This title is generated by a display command; for example, the following command displays a column titled COUNTRY: <code>PRINT COUNTRY</code>
ACROSSTITLE	This title consists of the field name from an ACROSS phrase in the request; for example, the following phrase displays an ACROSS group titled COUNTRY: <code>ACROSS COUNTRY</code>
ACROSSVALUE	This title consists of one of the values of a field referenced in an ACROSS phrase; for example, the phrase ACROSS COUNTRY displays a column titled: <code>LONDON</code>

The following request demonstrates each type of title. The numbers on the left refer to the annotated report output that follows the request:

```
TABLE FILE EMPLOYEE
1. SUM GROSS DED_AMT
2. ACROSS DEPARTMENT
1. BY HIGHEST PAY_DATE AS 'PAY DATE'
   WHERE PAY_DATE FROM 820131 TO 821231
   END
```

The numbers on the report output indicate which statements in the request produced each type of title:

```
PAGE      1
```

	DEPARTMENT			
	MIS	PRODUCTION		
PAY_DATE	GROSS	DED_AMT	GROSS	DED_AMT
82/08/31	\$9,000.00	\$4,575.76	\$9,523.84	\$4,911.14
82/07/30	\$7,460.00	\$4,117.07	\$7,048.84	\$3,483.89
82/06/30	\$7,460.00	\$4,117.07	\$7,048.84	\$3,483.89
82/05/28	\$6,649.51	\$3,954.39	\$7,048.84	\$3,483.89
82/04/30	\$5,890.84	\$3,386.76	\$4,959.84	\$2,061.70
82/03/31	\$3,247.75	\$1,740.88	\$4,959.84	\$2,061.70
82/02/26	\$3,247.75	\$1,740.88	\$4,959.84	\$2,061.70

Annotations for the report output:

- 1. TYPE=TITLE (points to the first column title, PAY_DATE)
- 2. TYPE=ACROSSTITLE (points to the DEPARTMENT header)
- 2. TYPE=ACROSSVALUE (points to the PRODUCTION header)

To select a type of title to format, use one of the following:

```
TYPE = TITLE
TYPE = ACROSSTITLE
TYPE = ACROSSVALUE
```

You can also select titles in specific columns. See *Selecting Report Columns* on page 10-34 for details.

Syntax

How to Select Column Totals

StyleSheet declarations can distinguish between three types of totals:

TYPE	Definition
GRANDTOTAL	The last total displayed in a report.
SUBTOTAL	Column totals or subtotals.
RECAP	Totals generated by RECAP or ON field COMPUTE calculations.

The following request demonstrates each type of total. The numbers on the left refer to the annotated report output that follows the request:

```
TABLE FILE EMPLOYEE
SUM GROSS DED_AMT AND
COMPUTE DG_RATIO/F4.2=DED_AMT/GROSS; AS 'RATIO'
BY DEPARTMENT
BY PAY_DATE
1. ON DEPARTMENT RECOMPUTE UNDER-LINE
2. ON DEPARTMENT RECAP DEPT_NET/D8.2M=GROSS-DED_AMT;
3. ON TABLE COLUMN-TOTAL
WHERE PAY_DATE FROM 820101 TO 820301
END
```

The numbers on the report output indicate which statements in the request produced each type of total:

PAGE 1

DEPARTMENT	PAY_DATE	GROSS	DED_AMT	RATIO	
MIS	02/01/29	\$3,247.75	\$1,740.88	.54	
	02/02/26	\$3,247.75	\$1,740.88	.54	
*TOTAL MIS		\$6,495.51	\$3,481.75	.54	← 1. TYPE=SUBTOTAL
** DEPT_NET		\$3,013.76			← 2. TYPE=RECAP
PRODUCTION	02/01/29	\$3,705.84	\$1,560.10	.42	
	02/02/26	\$4,959.84	\$2,061.70	.42	
*TOTAL PRODUCTION		\$8,665.68	\$3,621.81	.42	← 1. TYPE=SUBTOTAL
** DEPT_NET		\$5,043.87			← 2. TYPE=RECAP
TOTAL		\$15,161.18	\$7,103.56	.47	← 3. TYPE=GRANDTOTAL

To select a type of total to format, the syntax is:

```
TYPE = tottype
```

where:

tottype

Can be SUBTOTAL, RECAP or GRANDTOTAL.

You can select a specific subtotal or recap line by identifying the sort field that generated it. The syntax is

```
TYPE = {SUBTOTAL|RECAP}    [,BY =    {Bn|fieldname}]
```

where:

Bn

Identifies a BY field by its position in the report; *n* is a positive integer that counts all BY fields in the report from left to right, including NOPRINT BY fields.

fieldname

Identifies a BY field by its field name.

If a field appears in the report more than once, *fieldname(n)* selects the *nth* occurrence, and *fieldname(*)* selects all occurrences of the field.

You can also select totals in specific columns. See *Selecting Report Columns* on page 10-34 for details.

Selecting Headings and Footings

StyleSheet declarations can select the following types of headings and footings:

TYPE	Definition
TABHEADING	The report heading.
TABFOOTING	The report footing.
HEADING	The page heading.
FOOTING	The page footing.
SUBHEAD	A sort heading.
SUBFOOT	A sort footing.

The following request demonstrates most of the heading and footing report components. The numbers on the left refer to the annotated report output that follows the request:

```
TABLE FILE EMPLOYEE
1.  ON TABLE SUBHEAD
1.  "CONFIDENTIAL INFORMATION"
1.  "SWIFTY INFORMATION GROUP - EMPLOYEE LIST BY DEPARTMENT </1"
2.  HEADING CENTER
2.  "EMPLOYEES FOR DEPARTMENT: <DEPARTMENT </1"
   PRINT CURR_SAL HIRE_DATE
   BY DEPARTMENT NOPRINT
3.  BY JOB_DESC NOPRINT SUBHEAD
3.  "</1 <JOB_DESC ..."
   BY LAST_NAME
4.  BY FIRST_NAME SUBFOOT
4.  "*** REVIEW SALARY FOR <FIRST <LAST"
   WHEN CURR_SAL LT 18000
   WHERE RECORDLIMIT EQ 3
END
```

The numbers on the report output indicate which statements in the request produced each type of heading or footing:

PAGE1

CONFIDENTIAL INFORMATION
SWIFTY INFORMATION GROUP - EMPLOYEE LIST BY DEPARTMENT

EMPLOYEES FOR DEPARTMENT: MIS

LAST_NAME

FILE QUALITY ...

SMITH

** REVIEW SALARY FOR MARY SMITH

SECRETARY ...

STEVENS

FIRST_NAME

MARY

FOR MARY SMITH

ALFRED

CURR_SAL

\$13,200.00

\$11,000.00

\$11,000.00

HIRE_DATE

81/07/01

80/06/02

80/06/02

1. TYPE=PAGENUM

1. TYPE=TABHEADING

2. TYPE=HEADING

3. TYPE=SUBHEAD

4. TYPE=SUBFOOT

3. TYPE=SUBHEAD

Syntax

How to Select Headings and Footings

The following syntax selects a type of heading or footing:

```
TYPE = headtype
```

where:

headtype

Can be one of the following: TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD or SUBFOOT.

You can select a specific SUBHEAD or SUBFOOT by identifying the sort field that generated it. The syntax is

```
TYPE = {SUBHEAD|SUBFOOT} [,BY = {Bn|fieldname}]
```

where:

Bn

Identifies a BY field by its position in the report; *n* is a positive integer that counts all BY fields in the report from left to right, including NOPRINT BY fields.

fieldname

Identifies a BY field by its field name.

If a field appears in the report more than once, *fieldname(n)* selects the *n*th occurrence, and *fieldname(*)* selects all occurrences of the field.

In addition to the heading or footing text, headings and footings can include database fields, DEFINE fields, and page numbers created by the TABPAGENO field. You can select specific elements of a heading or footing with the following selection attributes (qualifiers):

Attribute	Definition
LINE	Selects a particular line in a multi-line heading or footing.
OBJECT	Selects a type of subcomponent: text phrases or embedded fields.
ITEM	Selects a particular text phrase or field.

The syntax for selecting a subcomponent of a heading or footing is

```
TYPE = {headtype [BY={Bn|fieldname}][LINE=n][ ,OBJECT={TEXT|FIELD}][ITEM=m]
```

where:

headtype

Can be one of the following: TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD or SUBFOOT.

n

Selects a specific line within a multi-line heading or footing, counting blank lines specified by “ ” and lines skipped with </n. For example:

HEADING

1. "The <CAR <MODEL is made in <COUNTRY and"
2. "sells for \$<RCOST (as of &DATE)."
3. " "
4. "Number of seats: <SEATS"

TEXT|FIELD

Selects either the text phrases or the embedded fields in a heading or footing. FIELD selects only data source fields and DEFINE fields, not amper variables. For example, in the following HEADING phrase, &DATE is TEXT; DEPT is a FIELD:

HEADING CENTER

"Report created on &DATE for <DEPT"

You can use the OBJECT attribute with LINE and/or ITEM attributes to select specific text phrases or fields.

Note: Spot markers split headings and footings into multiple parts. You can deliberately split headings and footings using <+0>.

m

Selects a part of a heading or footing by its position in the heading or footing:

- If used without an OBJECT attribute, ITEM counts all text phrases and fields, starting from 1 on each line. The following selects the second item on the line, regardless of whether that item is a text phrase or a field:

ITEM=2

- In a heading or footing with multiple text phrases, you can select a specific phrase. Count text phrases from left to right starting from 1 on each line. Text phrases are delimited by embedded fields or spot markers. The following selects the third text phrase:

ITEM=3, OBJECT=TEXT

- In a heading or footing with multiple fields, you can select a specific field. Count fields from left to right starting from 1 on each line. The following selects the second field:

ITEM=2, OBJECT=FIELD

- If the whole heading or footing is text with no embedded fields, the text counts as one item. You can break it into multiple items with spot markers. Consider the following HEADING phrase:

HEADING

"Report Created on <+0 &DATE <+0 for <DEPT"

You can select the ampersand variable, &DATE, with

ITEM=2, OBJECT=TEXT

BY

Is described *Selecting Headings and Footings* on page 10-28.

The following table illustrates selection attributes:

Attribute	Definition
OBJECT=TEXT	Selects all text in a heading or footing for formatting. Use the OBJECT=TEXT attribute by itself if you want all text phrases in a heading or footing to have the same style.
OBJECT=FIELD	Selects all embedded fields in a heading or footing for formatting. Use the OBJECT=FIELD attribute by itself if you want all fields in a heading or footing to have the same style.
OBJECT=TEXT, ITEM= <i>n</i>	Selects a specific text phrase for formatting. Text phrases are counted from left to right in a heading or footing line.
OBJECT=FIELD, ITEM= <i>n</i>	Selects a specific field for formatting. Fields are counted from left to right in a heading or footing line.
LINE= <i>n</i> , OBJECT=TEXT	Selects all text phrases on a specified heading or footing line.
LINE= <i>n</i> , OBJECT=FIELD	Selects all embedded fields on a specified heading or footing line.
LINE= <i>n</i> , ITEM= <i>m</i>	Selects the <i>m</i> th item on the <i>n</i> th line of a heading or footing, regardless of whether this item is a text phrase or an embedded field.
LINE= <i>n</i> , OBJECT=TEXT, ITEM= <i>m</i>	Selects the <i>m</i> th text phrase on the <i>n</i> th line of a heading or footing.
LINE= <i>n</i> , OBJECT=FIELD, ITEM= <i>m</i>	Selects the <i>m</i> th field on the <i>n</i> th line of a heading or footing.

The key to defining styles for subcomponents within a report heading or footing is knowing how to count the lines, text phrases, and fields within a heading or footing.

Example

Styling Heading and Footing Subcomponents

This example illustrates how to count and build styles for one or more lines, fields, and text phrases in a heading or footing:

```
TABLE FILE CAR
HEADING
1. "The <CAR <MODEL is made in <COUNTRY and"
2. "sells for $<RETAIL_COST (as of &DATE)."
   " "
3. "Number of Seats: <SEATS Weight: <WEIGHT "
   " "
   " "
PRINT BODYTYPE NOPRINT
FOOTING
4. "Supplied by"
   " "
   " The National Automobile Statistics Organization"
   " "
WHERE COUNTRY EQ 'ENGLAND';
ON TABLE SET STYLE HEAD
ON TABLE HOLD AS HEAD FORMAT PS
END
```

1. The first line of the report heading consists of three embedded fields (CAR, MODEL, and COUNTRY), and three text phrases ('The ', ' is made in ', and ' and').

The following declaration defines a style for all embedded fields in the first line:

```
TYPE=HEADING, LINE=1, OBJECT=FIELD, FONT=TIMES, SIZE=16,
STYLE=NORMAL, COLOR=BLACK , $
```

The next declaration defines a style for the second field, MODEL, in the first line:

```
TYPE=HEADING, LINE=1, OBJECT=FIELD, ITEM=2, FONT=TIMES,
SIZE=10,STYLE=BOLD, COLOR=BLACK, $
```

2. The second line of the report heading consists of one field (RETAIL_COST), and two text phrases ('sells for \$' and '(as of &DATE).').

The following declaration defines a style for the field RETAIL_COST:

```
TYPE=HEADING, LINE=2, OBJECT=FIELD, FONT=TIMES, SIZE=10, STYLE=BOLD,
COLOR=BLACK, $
```

The next declaration defines a style for the two text phrases:

```
TYPE=HEADING, LINE=2, OBJECT=TEXT, FONT=TIMES, SIZE=10, STYLE=ITALIC,
COLOR=BLACK, $
```

The following declaration defines a style for the second text phrase ('(as of &DATE).'):

```
TYPE=HEADING, LINE=2, OBJECT=TEXT, ITEM=2, FONT=TIMES, SIZE=10,STYLE=
UNDERLINE, COLOR=BLACK, $
```

3. This is the fourth line of the report heading, because “ “ in the previous line of the heading specifies a blank line, and blank lines count. It consists of two fields (SEATS and WEIGHT) and two text phrases (‘Number of Seats:’ and ‘Weight:’).

The following declaration defines a style for the first field, SEATS:

```
TYPE=HEADING, LINE=4, OBJECT=FIELD, ITEM=1, FONT=HELVETICA,
SIZE=10,STYLE=NORMAL,$
```

This declaration enhances the word ‘Weight:’ in the fourth line of the report heading:

```
TYPE =HEADING, OBJECT=TEXT, LINE=4, ITEM=2, SIZE=16,$
```

4. The four-line footing consists of two text lines (‘Supplied by’ and ‘The National Automobile Statistics Organization’).

The following declaration defines a style for both text lines:

```
TYPE=FOOTING, ITEM=1, FONT=TIMES, SIZE=16, STYLE=BOLD,$
TYPE=FOOTING, LINE=3, FONT=TIMES, STYLE=BOLD,$
```

Qualifiers are not required since there are no other sections or details (such as embedded fields) in this footing.

The StyleSheet named HEAD, consisting of these declarations, produces the following report:

Page 1

THE JAGUAR V12XKE AUTO IS MADE IN ENGLAND AND
SELLS FOR \$ 8,878 (AS OF 06/03/99).

NUMBER OF SEATS: 4 WEIGHT: 3,435

SUPPLIED BY

THE NATIONAL AUTOMOBILE STATISTICS ORGANIZATION

Selecting Report Columns

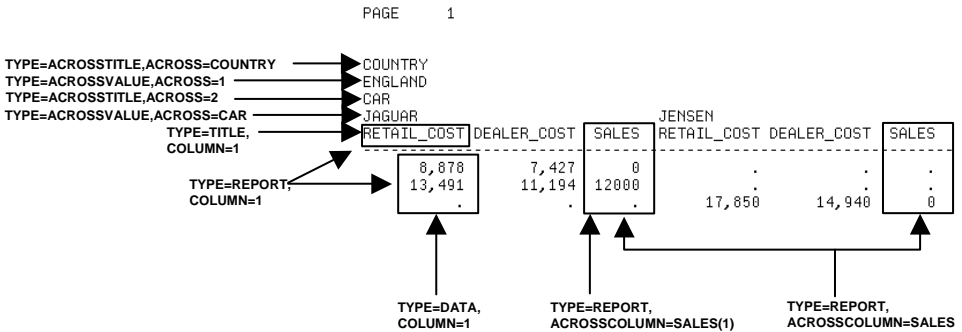
The following qualifiers select columns:

Qualifier	Selects the following subcomponent:
COLUMN	Columns in the report as a whole.
ACROSSCOLUMN	Columns in ACROSS groups.
ACROSS	A specific ACROSS field in a request with multiple ACROSS phrases.

Consider the following request that includes two ACROSS phrases (ACROSS COUNTRY and ACROSS CAR):

```
TABLE FILE CAR
PRINT RETAIL_COST DEALER_COST SALES
ACROSS COUNTRY
ACROSS CAR
WHERE COUNTRY EQ 'ENGLAND'
WHERE CAR EQ 'JAGUAR' OR 'JENSEN'
END
```

Since column attributes are subcomponents, you must use them in conjunction with a TYPE attribute that selects a report component. A subcomponent designates different parts of the report output depending on the TYPE attribute you specify with it; also, you can refer to the same column in several different ways. Some examples are marked on the following report output:



Syntax

How to Select Report Columns

The syntax is

TYPE = *coltype* [COLUMN={*colnotation* | ROWTOTAL[{*n*}|*fieldname*] }]

or

TYPE = *acrosscol* [ACROSSCOLUMN = {*acrosnotation*}]

or

TYPE = {ACROSSVALUE|ACROSSTITLE} [ACROSS = {*n*|*fieldname*}]

where:

coltype

Can be REPORT, TITLE, ACROSSVALUE, DATA, SUBTOTAL, or GRANDTOTAL.

colnotation

Can be *n*, *Pn*, *Cn*, *Bn*, or *fieldname*[(*n*)].

acrosscol

Can be REPORT, TITLE, DATA, SUBTOTAL, or GRANDTOTAL.

acrosnotation

Can be *n*, *Pn*, *fieldname*[(*n*)].

n

Is a positive integer that identifies a column by its position as follows:

Qualifier	Identifies position in . . .
COLUMN	The report. Count ACROSS fields, BY fields, NOPRINT fields, display fields, and ROW-TOTAL fields from left to right.
ACROSSCOLUMN	ACROSS groups counting from left to right.
ACROSS	The request. Count ACROSS phrases from top to bottom.

Pn

Is the same as *n*, except that it does not count NOPRINT fields.

Cn

Identifies a verb object column by its position from left to right in the report. Counts all fields, including NOPRINT fields, but excludes BY fields.

Note: C* selects all verb object columns in a report.

Bn

Identifies a BY field by its position from left to right in the report. Counts only BY fields, including NOPRINT BY fields.

Note: B* selects all BY fields in a report.

fieldname

Identifies a subcomponent by its field name as follows:

When used with:	Identifies:
<i>COLUMN</i>	A column. When a field appears more than once, use <i>fieldname(n)</i> to select a particular occurrence, or <i>fieldname(*)</i> to select all occurrences. (Note that <i>fieldname</i> is equivalent to <i>fieldname(1)</i>).
<i>ACROSSCOLUMN</i>	A column in ACROSS groups. When a field appears more than once, use <i>fieldname(n)</i> to select a particular occurrence.
<i>ACROSS</i>	A set of ACROSS titles or data.

ROWTOTAL

Identifies a row total column generated by ROW-TOTAL. When used with ACROSS, ROW-TOTAL may generate multiple total columns. ROWTOTAL(*n*) selects a particular total column. ROWTOTAL(*fieldname*) selects the row total column for a particular field. ROWTOTAL(*) selects all row total columns in the report.

Example**Styling With ACROSS Phrases**

The following report request contains two ACROSS phrases:

```
TABLE FILE CAR
SUM SALES
ACROSS COUNTRY
ACROSS CAR
ON TABLE SET STYLE SAMPLE3
ON TABLE HOLD AS SAMPLE3 FORMAT PS
END
```

Consider the SAMPLE3 StyleSheet:

```
TOPMARGIN = 0.125 , $
LEFTMARGIN = 0.125 , $
RIGHTMARGIN = 0.125 , $
TYPE=ACROSSTITLE, ACROSS=COUNTRY, STYLE=BOLD , $
TYPE=ACROSSVALUE, ACROSS=COUNTRY, STYLE=ITALIC , $
```

The report request and the StyleSheet produce the following report:

PAGE 1.1

COUNTRY			
<i>ENGLAND</i>	<i>ENGLAND</i>	<i>ENGLAND</i>	<i>FRANCE</i>
CAR			
JAGUAR	JENSEN	TRIUMPH	PEUGEOT
12000	0	0	0

Note:

- The word COUNTRY in the report is the title produced by the phrase ACROSS COUNTRY in the request. You select it with TYPE=ACROSSTITLE, ACROSS=COUNTRY.
- The names of the countries are the values of the COUNTRY field produced by the phrase ACROSS COUNTRY in the request. You select them with TYPE=ACROSSVALUE, ACROSS=COUNTRY.
- If ACROSS COUNTRY were the only ACROSS phrase in the request, you could omit the ACROSS qualifiers in the StyleSheet.

Positioning Headings, Footings, and Columns

The POSITION attribute defines a starting position for a column, a heading, or a footing.

Syntax

How to Position a Column

The syntax for positioning a column is

```
TYPE=REPORT, COLUMN = colnotation, POSITION = {n|+n}
```

where:

colnotation

Specifies a column (see *Selecting Report Columns* on page 10-34). Note that the column must be defined for TYPE=REPORT, and that it cannot be an ACROSSCOLUMN.

n

Is the amount of blank space, in UNITS, between the left margin and the beginning of the column.

This type of placement is called absolute positioning. It is easy to overlap columns using this method of column placement. For example, if you position a column at 1 inch from the margin and another column at 1.1 inch from the margin, the two columns will overlap if the first column is wider than 0.1 inch.

+*n*

Is the amount of blank space, in UNITS, between the end of the previous column and the beginning of this column.

This type of placement is called relative positioning.

Note: You can also specify column positions in a report request via spot markers and IN syntax; in this case the integer value in a column position refers to the number of spaces (measured in 12-point COURIER font) from the left side of the page. However, the StyleSheet POSITION attribute is the recommended method for positioning columns and heading elements.

Syntax

How to Position Headings and Footings

Headings and footings have an additional placement option. You can align them with a particular column. The syntax is

```
TYPE = headtype[element,] POSITION = {colnotation|[+}n}
```

where:

headtype

Can be one of the following: TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD or SUBFOOT.

element

Specifies a subcomponent of the heading or footing (see *Selecting Headings and Footings* on page 10-28).

colnotation

Specifies a column (see *Selecting Report Columns* on page 10-34). The heading or footing element is placed where the specified column starts. FIELD elements are justified according to the justification of the column; TEXT elements are aligned with the left edge of the column.

n

Is the amount of blank space, in UNITS, between the left margin and the beginning of the column.

+n

Is the amount of blank space, in UNITS, between the previously printed item and the beginning of this element or component.

Determining Column Widths

The SQUEEZE attribute assigns column widths. See *Styling the Page Layout* on page 10-13 for a description.

Changing Column Sequence

The SEQUENCE attribute defines the order in which columns are displayed on a report. By default, FOCUS prints BY fields first, then verb object fields in the order in which the request specifies them. For multi-verb requests, FOCUS prints the BY fields for the first verb, followed by the verb object fields of the first verb, followed by the remaining BY fields for the second verb, and the verb object fields for the second verb. For example:

```
TABLE FILE EMPLOYEE
SUM ED_HRS CURR_SAL
BY EMP_ID
PRINT ED_HRS CURR_SAL
BY EMP_ID
BY LAST_NAME
BY FIRST_NAME
END
```

The order of fields in the resulting report is EMP_ID, sum of ED_HRS, sum of CURR_SAL, LAST_NAME, FIRST_NAME, ED_HRS, CURR_SAL.

To change the sequence of a column in the report, select the column and define its sequence number.

Syntax

How to Change Column Sequence

The syntax is

```
TYPE = REPORT, COLUMN = colnotation, SEQUENCE = nnn
```

where:

colnotation

Specifies a column (see *Selecting Report Columns* on page 10-34).

nnn

Is a positive integer that represents the column's order in the report.

Note:

- SEQUENCE is valid only with TYPE=REPORT and COLUMN=colnotation. It reorders all information associated with a column: title, data, subtotal, and grandtotal.
- SEQUENCE is not valid with ACROSSCOLUMN or with any report output sorted across the page.
- SEQUENCE is not valid with fields that are displayed over each other using the OVER parameter. This includes the field immediately before the first OVER parameter.
- The sequence numbers you assign need not be in sequential order or in increments of one. FOCUS arranges the columns in lowest to highest order of the numbers you assign.

FOCUS assigns a number to each column in a report, starting with one on the far left. In the following request, the default column order is LAST_NAME, FIRST_NAME, EMP_ID, DEPARTMENT:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND DEPARTMENT
BY LAST_NAME
BY FIRST_NAME
END
```

When activated for the request, the StyleSheet named SEQU places EMP_ID in the second column. The revised request and the SEQU StyleSheet follow:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND DEPARTMENT
BY LAST_NAME
BY FIRST_NAME
ON TABLE SET STYLE SEQU
ON TABLE HOLD AS SEQU FORMAT PS
END
```

StyleSheet SEQU:

```
TYPE=REPORT, COLUMN=EMP_ID, SEQUENCE=2, FONT=TIMES, $
```

In the resulting stylized report output, FOCUS shifts columns two and three (FIRST_NAME and DEPARTMENT) one column to the right so it can place EMP_ID in column two:

PAGE 1

<u>LAST_NAME</u>	<u>EMP_ID</u>	<u>FIRST_NAME</u>	<u>DEPARTMENT</u>
BANNING	119329144	JOHN	PRODUCTION
BLACKWOOD	326179357	ROSEMARIE	MIS
CROSS	818692173	BARBARA	MIS
GREENSPAN	543729165	MARY	MIS
IRVING	123764317	JOAN	PRODUCTION
JONES	117593129	DIANE	MIS
MCCOY	219984371	JOHN	MIS
MCKNIGHT	451123478	ROGER	PRODUCTION
ROMANS	126724188	ANTHONY	PRODUCTION
SMITH	112847612	MARY	MIS
	119265415	RICHARD	PRODUCTION
STEVENS	071382660	ALFRED	PRODUCTION

Consider the following StyleSheet:

```
TYPE=REPORT, COLUMN=DEPARTMENT, SEQUENCE=999, FONT=TIMES, $
```

With this StyleSheet and any request that prints the field DEPARTMENT, DEPARTMENT is always the last column in the stylized report output. To ensure that a column appears last in the report output, select a sequence number that is greater than the total number of columns.

Specifying Column Spacing

The LEFTGAP and RIGHTGAP attributes define how much space to leave between columns.

Syntax

How to Specify Column Spacing

The syntax is

```
TYPE=REPORT,[COLUMN=colnotation,] {RIGHTGAP|LEFTGAP} = n
```

where:

colnotation

Specifies a column (see *Selecting Report Columns* on page 10-34).

n

Is the amount of blank space, in UNITS, to the right or left of the column area.

LEFTGAP and RIGHTGAP are only valid with TYPE=REPORT and, optionally, with the qualifier COLUMN.

Consider the following TABLE request:

```
SET STYLE = SPACING
TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME
BY DEPARTMENT
ON TABLE HOLD AS SPACING FORMAT PS
END
```

StyleSheet SPACING follows:

```
TYPE=REPORT, FONT=TIMES, SIZE=10 , $
TYPE=REPORT, COLUMN=EMP_ID, RIGHTGAP=1 , $
```

In the resulting report, the RIGHTGAP attribute places one additional inch of space to the right of the EMP_ID column:

PAGE 1

DEPARTMENT	EMP_ID	FIRST_NAME	LAST_NAME
MIS	112847612	MARY	SMITH
	117593129	DIANE	JONES
	219984371	JOHN	MCCOY
	326179357	ROSEMARIE	BLACKWOOD
	543729165	MARY	GREENSPAN
PRODUCTION	818692173	BARBARA	CROSS
	071382660	ALFRED	STEVENS
	119265415	RICHARD	SMITH
	119329144	JOHN	BANNING
	123764317	JOAN	IRVING
	126724188	ANTHONY	ROMANS
	451123478	ROGER	MCKNIGHT

The following request and StyleSheet place one inch before *every* column:

```
SET STYLE=RPTSPAC
TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME
BY DEPARTMENT
ONTABLE HOLD AS RPTSPAC FORMAT PS
END
TYPE=REPORT, FONT=TIMES, SIZE=10, LEFTGAP=1,$
```

Notice that the first column, DEPARTMENT, begins one inch from the left margin:

PAGE 1.1

DEPARTMENT	EMP_ID	FIRST_NAME
MIS	112847612	MARY
	117593129	DIANE
	219984371	JOHN
	326179357	ROSEMARIE
	543729165	MARY
PRODUCTION	818692173	BARBARA
	071382660	ALFRED
	119265415	RICHARD
	119329144	JOHN
	123764317	JOAN
	126724188	ANTHONY
	451123478	ROGER

StyleSheet Inheritance

System defaults define the style of a report component until you create a style that changes one or more of its characteristics. For example, the font for every report component is COURIER until you define a style that changes that font.

If a style listed in the StyleSheet lacks one or more style attributes (FONT, SIZE, STYLE, and COLOR), the components formatted by that style inherit values for the omitted attributes.

Report components automatically inherit characteristics from larger components as follows:

- The entire report inherits its characteristics from the default style.
- All report components inherit their characteristics from the values defined for the entire report.
- Elements in a report component inherit their characteristics from the component to which they belong. For example, a particular line in a multi-line heading inherits the characteristics of that heading.

Example

Inheriting Styles

Consider the following example:

```
SET ORIENTATION = LANDSCAPE
SET SQUEEZE = ON

TABLE FILE EMPLOYEE
SUM AVE.GROSS AS 'AVERAGE,GROSS' AND GROSS AS 'GROSS,AMOUNT' AND
COMPUTE NET_PAY/D12.2 = GROSS - DED_AMT; AS 'NET,PAY'
BY HIGHEST PAY_DATE AS 'PAY DATE'
ACROSS DEPARTMENT
HEADING
"Swifty Information Group"
"Department Payroll Report"
" "
"Sorted down by Pay Date, (most recent first)"
"and across the page by Department"
" "
FOOTING
"Confidential Information "
IF DEPARTMENT IS MIS OR PRODUCTION
ON TABLE SET STYLE ACROSS
ON TABLE HOLD AS ACROSS FORMAT PS
END
```

StyleSheet ACROSS:

```
TYPE=REPORT, FONT=TIMES, SIZE=10,$
TYPE=HEADING, SIZE=16, STYLE=BOLD,$
TYPE=HEADING, LINE=4, SIZE=14, STYLE=ITALIC,$
TYPE=HEADING, LINE=5, SIZE=14, STYLE=ITALIC,$
1. TYPE=ACROSSTITLE, SIZE=12, STYLE=UNDERLINE,$
2. TYPE=ACROSSVALUE, STYLE=ITALIC, FONT=HELVETICA,$
3. TYPE=TITLE, FONT=HELVETICA, SIZE=8,$
4. TYPE=DATA, ACROSSCOLUMN=4, FONT=HELVETICA, SIZE=8,$
```

In the resulting report:

1. The ACROSSTITLE is DEPARTMENT, generated by the phrase ACROSS DEPARTMENT in the report request.

The declaration for TYPE=ACROSSTITLE makes its size 12 points and its text style UNDERLINE. It inherits its font (TIMES) from TYPE=REPORT and its color (BLACK) from system defaults.

2. The ACROSS values are MIS and PRODUCTION, generated by the phrase ACROSS DEPARTMENT in the report request.

The declaration for TYPE=ACROSSVALUE in the StyleSheet formats their font as HELVETICA and their text style as ITALIC. They inherit their size (10 points) from TYPE=REPORT and their color (BLACK) from system defaults.

- The column titles under each ACROSSVALUE are 'AVERAGE GROSS', 'GROSS,AMOUNT', and 'NET,PAY'. There is one set of these titles under MIS and one set under PRODUCTION in the report. However, since the COMPUTE expression references DED_AMT, FOCUS adds DED_AMT as a NOPRINT field in each group between GROSS and NET_PAY. Therefore, NET_PAY becomes the fourth field.

The declaration for TYPE=TITLE makes the font for column titles HELVETICA and their size 8 points. They inherit their text style (NORMAL) and their color (BLACK) from system defaults.

- The declaration for TYPE=DATA, ACROSSCOLUMN=4, refers to the data values in the fourth column (the NET_PAY field; see item 3) under each ACROSSVALUE. This declaration makes their font HELVETICA and their size 8 points. They inherit their text style (NORMAL) and their color (BLACK) from system defaults.

The FOOTING inherits its style from TYPE=REPORT.

The report output (sized to fit) follows:

PAGE 1

Swiftly Information Group Department Payroll Report

*Sorted down by Pay Date, (most recent first)
and across the page by Department*

PAY DATE	<u>DEPARTMENT</u>					
	<i>MIS</i>			<i>PRODUCTION</i>		
	AVERAGE GROSS	GROSS AMOUNT	NET PAY	AVERAGE GROSS	GROSS AMOUNT	NET PAY
82/08/31	\$1,500.00	\$9,000.00	4,424.24	\$1,587.31	\$9,523.84	4,612.70
82/07/30	\$1,492.00	\$7,460.00	3,342.93	\$1,409.77	\$7,048.84	3,564.95
82/06/30	\$1,492.00	\$7,460.00	3,342.93	\$1,409.77	\$7,048.84	3,564.95
82/05/28	\$1,662.38	\$6,649.51	2,695.11	\$1,409.77	\$7,048.84	3,564.95
82/04/30	\$1,472.71	\$5,890.84	2,504.08	\$1,239.96	\$4,959.84	2,898.13
82/03/31	\$1,623.88	\$3,247.75	1,506.88	\$1,239.96	\$4,959.84	2,898.13
82/02/26	\$1,623.88	\$3,247.75	1,506.88	\$1,239.96	\$4,959.84	2,898.13
82/01/29	\$1,623.88	\$3,247.75	1,506.88	\$1,235.28	\$3,705.84	2,145.73
81/12/31	\$2,147.75	\$2,147.75	740.97	\$833.33	\$833.33	691.67
81/11/30	\$2,147.75	\$2,147.75	740.97	\$833.33	\$833.33	691.67

Confidential Information

Example

Styling Subtotals and Grand Totals

The next example illustrates subtotals and grand total:

```
TABLE FILE EMPLOYEE
PRINT DED_AMT AS 'DEDUCTION'
BY EMP_ID AS 'EMPLOYEE'
BY LAST_NAME NOPRINT
BY FIRST_NAME NOPRINT
BY PAY_DATE AS 'PAY DATE'
BY DED_CODE AS 'DEDUCTION CODE'
ON EMP_ID SUBTOTAL
ON PAY_DATE SUBTOTAL
ON FIRST_NAME SUBHEAD
"<FIRST_NAME <HIRE_DATE "
HEADING
"EMPLOYEES WITH JOBCODE <CURR_JOBCODE "
" "
ON TABLE COLUMN-TOTAL
WHERE CURR_JOBCODE EQ 'B04';
WHERE PAY_DATE EQ 820730;
WHERE DED_CODE EQ 'HLTH' OR 'LIFE';
ON TABLE SET STYLE TOTALS
ON TABLE HOLD AS TOTALS FORMAT PS
END
```

StyleSheet TOTALS:

- ```
TYPE=REPORT, FONT=TIMES,$
1. TYPE=SUBTOTAL, BY=B1, STYLE=BOLD,$
2. TYPE=SUBTOTAL, BY=B4, STYLE=ITALIC,$
3. TYPE=GRANDTOTAL, SIZE=14,$
 TYPE=DATA, COLUMN=B1, FONT=TIMES, STYLE=BOLD,$
 TYPE=DATA, COLUMN=B4, FONT=TIMES, STYLE=BOLD,$
```

In the resulting report:

1. The first BY column is EMPLOYEE, generated by the phrase BY EMP\_ID AS 'EMPLOYEE' in the request.

The declaration for TYPE=SUBTOTAL, BY=B1 makes the text style for the subtotal line generated by the phrase ON EMP\_ID SUBTOTAL bold. This subtotal line inherits its font (TIMES) from TYPE=REPORT and its size and color (12 points and BLACK) from system defaults.

2. The fourth BY column is PAY DATE. It appears to be the second BY column because LAST\_NAME and FIRST\_NAME are NOPRINT fields.

The declaration for TYPE=SUBTOTAL, BY=B4 makes the text style for the subtotal line generated by the phrase ON PAY\_DATE SUBTOTAL italic. This subtotal line inherits its font (TIMES) from TYPE=REPORT and its size and color (12 points and BLACK) from system defaults.

3. The grand total is the last total in a report. In this case it is the total generated by the phrase ON TABLE COLUMN-TOTAL.

The declaration for TYPE=GRANDTOTAL makes the size of this total 14 points. It inherits its font (TIMES) from TYPE=REPORT and its style and color (NORMAL and BLACK) from system defaults.

Following is the report output:

PAGE 1

EMPLOYEES WITH JOBCODE B04

| <u>EMPLOYEE</u>                        | <u>PAY DATE</u> | <u>DEDUCTION CODE</u> | <u>DEDUCTION</u>   |
|----------------------------------------|-----------------|-----------------------|--------------------|
| ANTHONY 82/07/01<br><b>126724188</b>   | <b>82/07/30</b> | HLTH<br>LIFE          | \$26.40<br>\$15.84 |
| *TOTAL PAY_DATE 82/07/30               |                 |                       | \$42.24            |
| *TOTAL EMP_ID 126724188                |                 |                       | <b>\$42.24</b>     |
| ROSEMARIE 82/04/01<br><b>326179357</b> | <b>82/07/30</b> | HLTH<br>LIFE          | \$45.37<br>\$27.22 |
| *TOTAL PAY_DATE 82/07/30               |                 |                       | \$72.60            |
| *TOTAL EMP_ID 326179357                |                 |                       | <b>\$72.60</b>     |
| TOTAL                                  |                 |                       | \$114.84           |

Example Styling RECAP Lines

The final example illustrates RECAP lines:

```
TABLE FILE EMPLOYEE
SUM GROSS AS 'Gross Amount' AND DED_AMT AS 'Deduction Amt' AND
COMPUTE DG_RATIO/F4.2 = DED_AMT / GROSS; AS 'RATIO'
BY DEPARTMENT AS 'Department'
BY PAY_DATE AS 'Pay Date'
WHERE PAY_DATE EQ 820730
ON DEPARTMENT RECAP DEPT_NET/D8.2M=GROSS - DED_AMT;
ON PAY_DATE RECAP NET/D8.2M=GROSS - DED_AMT;
HEADING
" Pay Summary"
ON TABLE SET STYLE RECAP
ON TABLE HOLD AS RECAP FORMAT PS
END
```

StyleSheet RECAP.

```
TYPE=REPORT, FONT=TIMES, SIZE=10,$
1. TYPE=RECAP, BY=B2, STYLE=ITALIC,$
2. TYPE=RECAP, BY=B1, STYLE=BOLD,$
```

In the report:

- 1. The second BY field in the request is PAY\_DATE. Therefore, the declaration for TYPE=RECAP, BY=B2 refers to the line in the report generated by the phrase ON PAY\_DATE RECAP NET ... in the request. This declaration makes the NET line italic. It inherits its font and size (TIMES and 10 points) from TYPE=REPORT and its color (BLACK) from system defaults.
- 2. The first BY field is DEPARTMENT. Therefore, the declaration for TYPE=RECAP, BY=B1 refers to the line generated by the phrase ON DEPARTMENT RECAP DEPT\_NET ... in the request. This declaration makes the DEPT\_NET line bold. It inherits its font and size (TIMES and 10 points) from TYPE=REPORT and its color (BLACK) from system defaults.

| PAGE 1                    |          |              |               |       |
|---------------------------|----------|--------------|---------------|-------|
| Pay Summary<br>Department | Pay Date | Gross Amount | Deduction Amt | RATIO |
| MIS                       | 82/07/30 | \$7,460.00   | \$4,117.07    | .55   |
| ** NET                    |          | 3,342.93     |               |       |
| ** DEPT_NET               |          | 3,342.93     |               |       |
| PRODUCTION                | 82/07/30 | \$7,048.84   | \$3,483.89    | .49   |
| ** NET                    |          | 3,564.95     |               |       |
| ** DEPT_NET               |          | 3,564.95     |               |       |

# Conditional Styling

Conditional styling, or stoplighting, formats a report component based on the results of a conditional test. The WHEN attribute specifies the test condition. The syntax is

```
WHEN = column operator {value|column}
```

where:

*column*

Specifies a single column in the report (see *Selecting Report Columns* on page 10-34). You can include prefixes such as TOT. or MIN. in front of a field name as long as the prefixed fields appear in the TABLE request.

*operator*

Can be one of the following: EQ, NE, LE, LT, GE or GT.

*value*

Is a number or a string enclosed in single quotation marks. If it is a string, its case (for example, uppercase) must match the case of the data in the data source.

## Note:

- FOCUS ignores WHEN conditions that refer to a column that is not referenced in the report request. If you want to base a WHEN condition on a field that you do not want to print in the report, make the field a NOPRINT field. For example:

```
TABLE FILE CAR
SUM MAX.SALES NOPRINT
SUM SALES BY COUNTRY BY CAR
END
TYPE=HEADING, STYLE=BOLD, WHEN= MAX.SALES LT 10000 , $
```

- Arithmetical and logical expressions are not allowed on either side of the operator. However, you can create a temporary field that evaluates the expression and use that temporary field in the WHEN expression. For example:

```
DEFINE FILE CAR
TEST/I2 = RETAIL_COST GT (1.1 * DEALER_COST);
END
TABLE FILE CAR
.
.
.
END
TYPE=DATA, COLUMN=RETAIL_COST , COLOR=RED, WHEN=TEST NE 0 , $
```

The field TEST must be a field in the report request. If you do not want to print it in the report, make it a NOPRINT field.

- If more than one WHEN condition applies to a report component, FOCUS evaluates them in the order in which they appear in the StyleSheet. For example:

```
TYPE=DATA, COLOR=RED, WHEN=RETAIL_COST GT 12000, $
TYPE=DATA, COLOR=GREEN, WHEN=RETAIL_COST GT 8000, $
```

Any data line in which the value of RETAIL\_COST satisfies both conditions is printed in red because the WHEN condition for red comes first in the StyleSheet. You can use this feature to implement simple if-then-else logic. You can implement more complex logic by combining this technique with the use of temporary fields.

## Example

### Conditional Styling

The following StyleSheet prints lines in red and bold when the value of the field SALES is greater than \$1,000:

```
TYPE=DATA, STYLE=BOLD, COLOR=RED, WHEN=SALES GT 1000, $
```

The next StyleSheet prints the value in the column SALES in red if SALES is less than \$100,000, or in green if SALES exceeds \$200,000:

```
TYPE=DATA, COLUMN=SALES, COLOR=RED, WHEN=SALES LT 100000, $
TYPE=DATA, COLUMN=SALES, COLOR=GREEN, WHEN=SALES GT 200000, $
```

This StyleSheet prints the data value in column C1 in bold whenever the value of SALES is greater than \$1,000:

```
TYPE=DATA, COLUMN=C1, STYLE=BOLD, WHEN=SALES GT 1000, $
```

In the next example, all lines representing instances in which COUNTRY has the value ENGLAND are shown in bold. If COUNTRY is a sort (BY) field, the entire sort group is shown in bold, even though the value ENGLAND appears on only one line:

```
TYPE=DATA, WHEN=COUNTRY EQ 'ENGLAND', STYLE=BOLD, $
```

The next example illustrates that for TYPE=REPORT, an entire report column—including the title, data, subtotal, and grand total—can change if the WHEN condition is based on an aggregate value:

```
TYPE=REPORT, COLUMN=C1, WHEN=TOT.SALES LT 100000, COLOR=RED, $
```

## Using WHEN With ACROSSCOLUMN

If you use WHEN with ACROSSCOLUMN, styles are applied differently depending on whether the column referenced in the WHEN condition falls within ACROSS groups. A WHEN column that is within an ACROSS group controls the formatting of all data within the same ACROSS group.

Consider the following report output:

|         | SEATS       |             |             |             |
|---------|-------------|-------------|-------------|-------------|
|         | 4           |             | 5           |             |
| COUNTRY | RETAIL_COST | DEALER_COST | RETAIL_COST | DEALER_COST |
| ENGLAND | 8000        | 7000        | 7500        | 6000        |
| JAPAN   | 9000        | 8000        | 10000       | 9000        |

In this report, both RETAIL\_COST and DEALER\_COST are printed ACROSS SEATS. However, COUNTRY does not fall within the ACROSS group.

In the following StyleSheet, data values in the RETAIL\_COST columns are formatted according to the data in their corresponding DEALER\_COST columns:

```
TYPE=DATA, ACROSSCOLUMN=RETAIL_COST, COLOR=RED, WHEN=DEALER_COST GT 7100,$
```

Therefore, the value 9000 in the RETAIL\_COST column under SEATS=4 is printed in red, since its corresponding DEALER\_COST value (8000) is greater than 7100; the RETAIL\_COST value 7500 is not printed in red because its corresponding DEALER\_COST is not greater than 7100.

In the next StyleSheet, the WHEN condition references the column COUNTRY, which is not part of an ACROSS group:

```
TYPE=DATA, ACROSSCOLUMN=RETAIL_COST, COLOR=RED, WHEN=COUNTRY EQ 'ENGLAND', $
```

In this case all RETAIL\_COST values in the line for ENGLAND are printed in red.

If a StyleSheet uses ACROSSCOLUMN with WHEN and a field name referenced in the WHEN condition appears both under the ACROSS and elsewhere in the report (as is possible with a multi-verb request), the field name under the ACROSS takes precedence. You can refer to the other column using another version of the column notation, such as Cn.

For example, in the next request, the RETAIL\_COST column under each value of CAR may be printed in bold depending on the corresponding value of DIFF for each CAR:

```
TABLE FILE CAR
SUM RETAIL_COST AND DEALER_COST
AND COMPUTE DIFF/D12.2=RETAIL_COST - DEALER_COST;
ACROSS CAR
ON TABLE SET SQUEEZE ON
ON TABLE SET STYLE 120193A
ON TABLE HOLD AS 120193A FORMAT PS
END
```

StyleSheet 120193A:

```
TYPE=REPORT, FONT=TIMES, SIZE=10,$
TYPE=DATA, ACROSSCOLUMN=RETAIL_COST, SIZE=14, WHEN=DIFF GT 9000,
STYLE=BOLD,$
```

Two pages of the resulting report follow:

PAGE 1.1

| CAR<br>ALFA ROMEO |             |          | AUDI        |             |        |
|-------------------|-------------|----------|-------------|-------------|--------|
| RETAIL_COST       | DEALER_COST | DIFF     | RETAIL_COST | DEALER_COST | DIFF   |
| 19,565            | 16,235      | 3,330.00 | 5,970       | 5,063       | 907.00 |

PAGE 1.2

| CAR<br>BMW    |             |          | DATSUN      |             |        |
|---------------|-------------|----------|-------------|-------------|--------|
| RETAIL_COST   | DEALER_COST | DIFF     | RETAIL_COST | DEALER_COST | DIFF   |
| <b>58,762</b> | 49,500      | 9,262.00 | 3,139       | 2,626       | 513.00 |

To specify the DIFF field outside the ACROSS, you can use the notation C3:

```
WHEN=C3 GT 9000
```



**Example****Conditional Styling With a BY Field**

The following example uses the notation COLUMN = B1 to select the COUNTRY column and to make it bold, italic, and 14-point for lines with SALES greater than 9000:

```
TABLE FILE CAR
PRINT CAR AND SALES
BY COUNTRY
HEADING
"USING STOPLIGHTING COLUMN=SALES, WHEN=SALES GT 9000"
ON TABLE SET STYLE STOPLIT4
ON TABLE HOLD AS STOPLIT4 FORMAT PS
END
```

StyleSheet STOPLIT4:

```
TYPE=REPORT, FONT=TIMES,$
TYPE=DATA, COLUMN=B1, SIZE=14, STYLE=BOLD+ITALIC, WHEN=SALES GT 9000,$
```

The report output follows:

PAGE 1

```
USING STOPLIGHTING COLUMN=SALES, WHEN=SALES GT 9000
COUNTRY CAR SALES

ENGLAND JAGUAR 0
 JAGUAR 12000
 JENSEN 0
 TRIUMPH 0
FRANCE PEUGEOT 0
ITALY ALFA ROMEO 12400
 ALFA ROMEO 13000
 ALFA ROMEO 4800
 MASERATI 0
JAPAN DATSUN 43000
 TOYOTA 35030
W GERMANY AUDI 7800
 BMW 8950
 BMW 8900
 BMW 14000
 BMW 18940
 BMW 14000
 BMW 15600
```

Example Conditional Styling With A NOPRINT Field

The next example uses the value of total sales in the WHEN condition. The column TOT.SALES must be referenced in the request or FOCUS ignores the WHEN condition. However, it is a NOPRINT field and is not printed in the report.

```
TABLE FILE CAR
SUM TOT.SALES NOPRINT
SUM SALES
BY COUNTRY
BY CAR
HEADING
"USING STOPLIGHTING"
"WHEN TOT.SALES GT 200000 HEADING LINE 2"
ON TABLE SET STYLE STOPLIT5
ON TABLE HOLD AS STOPLIT5 FORMAT PS
END
```

StyleSheet STOPLIT5:

```
TYPE=REPORT, FONT=TIMES, SIZE=10,$
TYPE=HEADING, LINE=2, SIZE=12, STYLE=BOLD,
 WHEN=TOT.SALES GT 200000,$
```

The report output follows:

|                                                |            |              |
|------------------------------------------------|------------|--------------|
| PAGE 1                                         |            |              |
| USING STOPLIGHTING                             |            |              |
| <b>WHEN TOT.SALES GT 200000 HEADING LINE 2</b> |            |              |
| <u>COUNTRY</u>                                 | <u>CAR</u> | <u>SALES</u> |
| ENGLAND                                        | JAGUAR     | 12000        |
|                                                | JENSEN     | 0            |
|                                                | TRIUMPH    | 0            |
| FRANCE                                         | PEUGEOT    | 0            |
| ITALY                                          | ALFA ROMEO | 30200        |
|                                                | MASERATI   | 0            |
| JAPAN                                          | DATSUN     | 43000        |
| W GERMANY                                      | TOYOTA     | 35030        |
|                                                | AUDI       | 7800         |
|                                                | BMW        | 80390        |

---

## CHAPTER 11

# Saving and Reusing Report Output

### Topics:

- Saving Your Report Output
- Creating HOLD and PCHOLD Files
- Holding Report Output in FOCUS Format
- Controlling Attributes in HOLD Master Files
- Keyed Retrieval From HOLD Files
- Creating SAVE and SAVB Files
- Choosing Output File Formats
- Saving Report Output in INTERNAL Format

When you run a report request, by default the data values that you request are collected and presented in a viewable form complete with column headings and other formatting features. You can take those same data values and, instead of formatting them for viewing, use them to create special data files to:

- Display as a Web page, as a printed document, or within a text document.
- Process in another application, such as a spreadsheet, a database, a word processor, or a 3GL program.
- Send to another location, such as a browser or PC.
- Extract a subset of the original data source in order to generate multi-step reports.

The output files you create are stored locally by FOCUS for S/390 and by WebFOCUS (Windows version) when used in a local environment.

The output files are stored on the server platform by WebFOCUS (browser version). When used in a server environment, the output files are also stored on the server platform by WebFOCUS (Windows version) and by FOCUS for S/390.

## Saving Your Report Output

The following commands enable you to extract and save report output in a variety of file types and formats to serve a wide range of purposes:

- **HOLD.** The HOLD command creates a data source containing the output of a report request. You can specify that the HOLD file's data be in BINARY format (the default), be formatted as a FOCUS data source, be formatted as a simple character file, or be held in a format suitable for use by a variety of other software products or helper applications. For some formats, the HOLD command also creates a corresponding Master File. You can then write other report requests that in turn extract or save data from the HOLD file. For details, see *Creating HOLD and PCHOLD Files* on page 11-3.
- **PCHOLD.** The PCHOLD command creates a data source containing the output of a report request, and downloads the HOLD data source and the optional Master File to a client computer or browser. As with a HOLD file, you can specify a variety of file formats. For details, see *Creating HOLD and PCHOLD Files* on page 11-3.

**Note:** If you specify an ON TABLE PCHOLD command without a FORMAT, XML/HTML code is returned to the browser. In WebFOCUS Version 4.2.1a, the Master File and report output are returned to the browser. To avoid this, use ON TABLE HOLD FORMAT ALPHA.

- **SAVE.** The SAVE command is identical to a HOLD command, except that it does not create a Master File, and the default format is ALPHA, not BINARY. If you wish to create a SAVE file in BINARY format, you can use a variation of the SAVE command called SAVB. For details, see *How to Create a SAVB File* on page 11-25.

As with a HOLD file, you can specify a variety of formats suitable for use with other software products. For details, see *Creating SAVE and SAVB Files* on page 11-23.

## Reference

### Naming and Preserving Report Output Files

Report output files remain usable until they are erased or written over. Subsequent output files created during a session replace the initial versions. Hence, only one output file of each type can be active at one time, unless you give it another name by using the AS phrase. For details see *How to Create a HOLD File* on page 11-4.

A FILEDEF or ALLOCATE command is automatically issued when you create an output file. The ddname used to identify the file is the same as the name of the report output file (HOLD, SAVE, or SAVB, or the name in the AS phrase), if not already allocated.

In addition, in the VM/CMS environment you can use a FILEDEF command to save an output file to a specific location and assign it a file name, file type, and file mode. In MVS, you can dynamically allocate an output file using the DYNAM ALLOCATE or TSO ALLOCATE command. See your *Overview and Operating Environments* manual for details.

# Creating HOLD and PCHOLD Files

You can use the HOLD and PCHOLD commands to create report output files for a range of purposes:

- As a tool for data extraction, the HOLD command enables you to retrieve and process data, then extract the results for further processing. That is, your report request can create a new data source, complete with a corresponding Master File for certain data formats, from which you can generate new reports.

The output Master File contains only the fields in the report request. The fields in a HOLD file have the original names that would be retrieved had the report been displayed or printed. You can alter the field names in the output Master File using the AS phrase in conjunction with the command SET ASNAMES. For details, see *Controlling Field Names in a HOLD Master File* on page 11-14.

When created in BINARY format (the default):

- The HOLD file is a sequential single-segment data source. The HOLD Master File is a subset of the original Master File, and may also contain fields that have been created using the COMPUTE or DEFINE commands or generated in an ACROSS phrase.
- By default fields with format I remain four-byte binary integers; format F fields remain in four-byte floating-point format; format D fields remain in eight-byte double-precision floating-point; and format P fields remain in packed decimal notation and occupy eight bytes (for fields less than or equal to eight-bytes long) or 16 bytes (for packed decimal fields longer than eight bytes). Alphanumeric fields (format A) are stored in character format.

Every data field in the sequential extract record is aligned on the start of a full 4-byte word. Therefore, if the format is A1, the field is padded with three bytes of blanks on the right. This alignment makes it easier for user-coded subroutines to process these data fields. (Under some circumstances, you may wish to prevent the padding of integer and packed decimal fields; you can do so with HOLD FORMAT INTERNAL. For details, see *Saving Report Output in INTERNAL Format* on page 11-42.)

- As a means of providing report output files for display or processing in other software applications, the HOLD command enables you to specify the appropriate formats. For details see *Choosing Output File Formats* on page 11-26.
- When an application requires a data format that is not among the HOLD options, you can use a subroutine to process each output record as it is written to the HOLD data source. For details see *How to Create a HOLD File* on page 11-4.

For information on writing programs to create HOLD files, see Appendix D, *Writing User-Coded Programs to Create HOLD Files*.

The PCHOLD command enables you to extract data from the WebFOCUS Reporting Server by way of the WebFOCUS client, and automatically display the data in HTML format in your browser.

In addition, if you have established a helper application in WebFOCUS or in the S/390 Web Interface, you can use the command ON TABLE PCHOLD to display the data in the helper application's viewer. For example, if a procedure contains the ON TABLE PCHOLD FORMAT EXCEL command, data is not returned to the browser in HTML format. Instead, data is returned and imported into an Excel spreadsheet, or other spreadsheet program you specify to your browser.

In contrast, when data access is handled directly by the iWay Server (without intervention by the WebFOCUS client), as is the case for WebFOCUS (Windows version) used in client/server mode and for FOCUS for S/390 when used as a client to iWay, then the data is extracted to a PCHOLD file and automatically delivered to your PC for local reporting.

**Note:** If your environment supports the SET parameter SAVEMATRIX, you can preserve the internal matrix of your last report in order to keep it available for subsequent HOLD, SAVE, and SAVB commands when the request is followed by Dialogue Manager commands. For details on SAVEMATRIX, see the *Developing Applications* manual.

## Syntax

### How to Create a HOLD File

You can create a HOLD file from a report request using the following syntax

```
ON TABLE HOLD [AS filename][FORMAT fmt][MISSING {ON|OFF}][VIA program]
```

or

```
hold_field HOLD [AS filename][FORMAT fmt][MISSING {ON|OFF}]
[VIA program]
```

After a report is executed, you can use the following syntax to create the HOLD file

```
HOLD [AS filename][FORMAT fmt][MISSING {ON|OFF}][VIA program]
```

where:

**HOLD**

Extracts and saves report output. When issued without an explicit format, the HOLD command uses its default format: BINARY. The output is saved with an associated Master File.

**hold\_field**

Is the name of the last field in the request.

**AS filename**

Specifies a name for the HOLD file. If you do not specify a file name, HOLD is used as the default name. Since each subsequent HOLD command overwrites the previous HOLD file, it is useful practice to code a distinct file name in each request to direct the extracted data to a separate file, thereby preventing it from being overwritten by the next HOLD command.

**FORMAT** *fmt*

Specifies the format of the HOLD output file. The default format is BINARY.

- To display as a Web page, choose:

HTML, HTMTABLE

These formats are supported through the Web Interface.

- To display as a printed document, choose:

PDF, PS

PDF format is supported through the Web Interface.

- To use in a text document, choose:

ALPHA, WP

- To use in a spreadsheet application, choose:

DIF, EXCEL, SYLK, LOTUS

- To use in a database application, choose:

FOCUS, DB2, FUSION, SQL, SQLORA

The following additional formats are supported when FOCUS is used as a client to iWay:

INGRES, REDBRICK, SQLDBC, SQLINF, SQLMSS, SQLSYB, SQLODBC

- To use with a 3GL program, choose:

INTERNAL

- To use for additional reporting in FOCUS, choose:

ALPHA, BINARY, FOCUS

- To use as a transaction file for modifying a data source, choose:

ALPHA, BINARY

For details about particular formats, see *Choosing Output File Formats* on page 11-26.

**MISSING**

Controls whether fields with the attribute MISSING=ON in the Master File are carried over into the HOLD file. MISSING ON is the default. If the HOLD command specifies MISSING OFF, the MISSING attribute is not carried over. For related information see Chapter 12, *Handling Records With Missing Field Values*.

**VIA** *program*

Calls a user-coded program to create the extract file. BINARY format is the default; ALPHA format is also available. Other formats are not available when you use a program with the HOLD command.

## **Syntax**

### **How to Create HOLD Files From Hot Screen**

You can specify a HOLD file from the FOCUS command line or from Hot Screen after the report is run and displayed.

The syntax is:

```
HOLD [AS filename][FORMAT fmt][MISSING {ON|OFF}][VIA program]
```

You can issue HOLD from the Hot Screen command line at any time while a report is displayed and on any page of the report. Regardless of the page from which you issue the command, the entire report is saved, and a data source and Master File are created for that report (just as they are when you issue the HOLD or PCHOLD command from within a TABLE request, or after exiting Hot Screen).

You can issue multiple HOLD commands for a single TABLE request; however, once you specify the FOCUS format with a HOLD command from Hot Screen, you cannot issue another HOLD command during that Hot Screen session.

Note that you cannot use the SAVE or SAVB commands from Hot Screen. You must include these commands in a report request, or issue them from the FOCUS command level after exiting Hot Screen.

## **Syntax**

### **How to Create a PCHOLD File**

The syntax for PCHOLD in a report request is

```
ON TABLE {PCHOLD|HOLD AT CLIENT}[AS filename] [FORMAT fmt]
```

where:

```
PCHOLD|HOLD AT CLIENT
```

Downloads HOLD files to a browser or other client application. HOLD AT CLIENT is a synonym for PCHOLD. When issued without an explicit format, the PCHOLD command uses its default format: ALPHA. The output is saved as character data with a Master File.

```
AS filename
```

Specifies a name for the PCHOLD file. If you do not specify a file name, HOLD is used as the default name. Since each subsequent PCHOLD command overwrites the previous PCHOLD file, it is useful practice to code a distinct file name in each request to direct the extracted data to a separate file, thereby preventing it from being overwritten by the next PCHOLD command.



`FORMAT fmt`

Specifies the format of the PCHOLD file. The default format is ALPHA.

- To display as or in a Web page, choose:

`HTML, HTMTABLE`

These formats are supported through the Web Interface.

- To display as a printed document, choose:

`PDF`

This format is supported through the Web Interface.

- To use in a text document, choose:

`ALPHA, WP`

- To use in a spreadsheet application, choose:

`DIF, EXCEL, LOTUS`

- To use for additional reporting in FOCUS, choose:

`ALPHA, BINARY`

For details about particular formats, see *Choosing Output File Formats* on page 11-26.

## Example

### Extracting Data to a HOLD File

The following request extracts data from the EMPLOYEE data source and creates a HOLD file:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AND ED_HRS
BY DEPARTMENT
LIST CURR_SAL AND ED_HRS AND BANK_ACCT
BY DEPARTMENT
BY LAST_NAME BY FIRST_NAME
ON TABLE HOLD
END
```

The following message displays:

```
NUMBER OF RECORDS IN TABLE= 12 LINES= 12
```

You will then see the message:

```
HOLDING...
```

To display the report generated by this request, you can issue a report request against the HOLD file. In FOCUS for S/390 and WebFOCUS (Windows version), you can also issue the RETYPE command.

#### Tip:

If you wish to view the information in the HOLD Master File before reporting against it, you can issue the query command ? HOLD. See *How to Query a HOLD Master File* on page 11-8.

Syntax

How to Query a HOLD Master File

If the HOLD format option you select creates a Master File, you can issue the command

```
? HOLD
```

to display the fields, aliases, and formats in the HOLD Master File. This command shows field names up to 32 characters. If a field name exceeds 32 characters, a caret (>) in the 32nd position indicates a longer field name.

If you have renamed the HOLD file using AS *filename*, use the following syntax:

```
? HOLD filename
```

Example

Reporting Against the HOLD Master File

In the following HOLD file, the formats shown are the values of the FORMAT attribute. You can see the values of the ACTUAL attribute by displaying the HOLD Master File using TED or another editor. USAGE and ACTUAL formats for text fields specify only the length of the first line of each logical record in the HOLD file. The USAGE format is the same as the field format in the original Master File. The ACTUAL format is rounded up to a full (internal) word boundary, as is done for alphanumeric fields.

The following request creates a HOLD file and displays the fields, aliases, and formats in the associated Master File:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AND ED_HRS
BY DEPARTMENT
LIST CURR_SAL AND ED_HRS AND BANK_ACCT
BY DEPARTMENT
BY LAST_NAME BY FIRST_NAME
ON TABLE HOLD
END
? HOLD
```

The output is:

| DEFINITION OF HOLD FILE: HOLD |       |        |
|-------------------------------|-------|--------|
| FIELDNAME                     | ALIAS | FORMAT |
| DEPARTMENT                    | E01   | A10    |
| CURR_SAL                      | E02   | D12.2M |
| ED_HRS                        | E03   | F6.2   |
| LAST_NAME                     | E04   | A15    |
| FIRST_NAME                    | E05   | A10    |
| LIST                          | E06   | I5     |
| CURR_SAL                      | E07   | D12.2M |
| ED_HRS                        | E08   | F6.2   |
| BANK_ACCT                     | E09   | I9S    |

You can now issue the following report request against the HOLD file:

```
TABLE FILE HOLD
PRINT E07 AS 'SALARY OF,EMPLOYEE' AND LAST_NAME AND FIRST_NAME
BY HIGHEST E03 AS 'TOTAL,DEPT,ED_HRS'
BY E01
BY HIGHEST E08 AS 'EMPLOYEE,ED_HRS'
END
```

The output is:

| TOTAL  |            |          |             |           |            |
|--------|------------|----------|-------------|-----------|------------|
| DEPT   |            | EMPLOYEE | SALARY OF   |           |            |
| ED_HRS | DEPARTMENT | ED_HRS   | EMPLOYEE    | LAST_NAME | FIRST_NAME |
| -----  | -----      | -----    | -----       | -----     | -----      |
| 231.00 | MIS        | 75.00    | \$21,780.00 | BLACKWOOD | ROSEMARIE  |
|        |            | 50.00    | \$18,480.00 | JONES     | DIANE      |
|        |            | 45.00    | \$27,062.00 | CROSS     | BARBARA    |
|        |            | 36.00    | \$13,200.00 | SMITH     | MARY       |
|        |            | 25.00    | \$9,000.00  | GREENSPAN | MARY       |
|        |            | .00      | \$18,480.00 | MCCOY     | JOHN       |
| 120.00 | PRODUCTION | 50.00    | \$16,100.00 | MCKNIGHT  | ROGER      |
|        |            | 30.00    | \$26,862.00 | IRVING    | JOAN       |
|        |            | 25.00    | \$11,000.00 | STEVENS   | ALFRED     |
|        |            | 10.00    | \$9,500.00  | SMITH     | RICHARD    |
|        |            | 5.00     | \$21,120.00 | ROMANS    | ANTHONY    |
|        |            | .00      | \$29,700.00 | BANNING   | JOHN       |

## Holding Report Output in FOCUS Format

Whether issued within a request or after the request has been executed, the HOLD command can create a FOCUS data source and a corresponding Master File using the data extracted by the report request. This feature enables you to create:

- A FOCUS data source from any other supported data source type.
- A subset of an existing FOCUS data source.

### Tip:

If you are working in an environment that supports SCAN, FSCAN, MODIFY, or Maintain, and you create a HOLD file in FOCUS format, you can update, as well as report against, the HOLD file. See your documentation on these facilities for details.

## Syntax

### How to Create HOLD Files in FOCUS Format

In a report request, the syntax is

```
ON TABLE HOLD [AS filename] FORMAT FOCUS [INDEX field1 field2 ...]
```

After a report request is run, the syntax is

```
HOLD [AS filename] FORMAT FOCUS [INDEX field1 field2 ...]
```

where:

*AS filename*

Specifies a name for the HOLD file. If you do not specify a file name, HOLD is used as the default name. Since each subsequent HOLD command overwrites the previous HOLD file, it is useful practice to code a distinct file name in each request to direct the extracted data to a separate file, thereby preventing it from being overwritten by the next HOLD command.

The name can be up to 64 characters long. If you use a name longer than eight characters on OS/390, an eight-character member name will be generated as described in the *Describing Data* manual. To relate the long name to the short member name, the \$ VIRT attribute will be generated on the top line in the Master File. The resulting HOLD file will be a temporary data file. To allocate the long Master File name to a permanent data file, issue the DYNAM ALLOCATE command with the LONGNAME option prior to the HOLD request. The ddname in the command must refer to an existing member of the MASTER PDS.

*INDEX field1...*

Enables you to index FOCUS fields. All fields specified after INDEX will be specified as FIELDTYPE=I in the Master File. Up to four fields can be indexed.

Note that once you use this format from Hot Screen, you cannot issue another HOLD command while in the same Hot Screen session.

## Reference

### Operating System Notes for HOLD Files in FOCUS Format

In CMS, a USE command is issued and the new data source and Master File are created on the disk that has WRITE permission and the most available space.

In MVS, the HOLD file is dynamically allocated if it is not currently allocated. This means the system may delete the file at the end of the session, even if you have not. Since HOLD files are usually deleted, this is a desired default; however, if you want to save the Master File, we recommend that it be allocated to ddname HOLDMAST as a permanent data set; the allocation can be performed in the standard FOCUS CLIST. For example,

```
ALLOC F(HOLDMAST) DA('qualif.HOLDMAST') SHR REUSE
```

Note that ddname HOLDMAST must not refer to the same PDS referred to by the MASTER and FOCEXEC ddnames.

## Reference

### Controlling the FOCUS File Structure

The structure of the FOCUS data source being created varies according to the report request. The rules are as follows:

- Each aggregation command (SUM, COUNT, WRITE) creates a segment, with each new BY field in the request becoming a key. In a request that uses multiple display commands, the key to any created segment does not contain keys that are in the parent segment.
- If a PRINT or LIST command is used to create a segment, all the BY fields, together with the internal FOCLIST field, form the key.
- All fields specified after INDEX will be indexed; that is, FIELDTYPE=I will be specified in the Master File. Up to four fields may be indexed.

To control whether the ACCEPT and TITLE attributes are propagated to the Master File associated with the HOLD file, use the SET HOLDATTR command. To control the FIELDNAME attribute in the Master of the HOLD file, use the SET ASNAMES command. For more information on how to control the TITLE, ACCEPT, and FIELDNAME attributes in a HOLD Master File see *Controlling Attributes in HOLD Master Files* on page 11-14.

#### Tip:

In environments that support the MODIFY facility, when the command HOLD FORMAT FOCUS is executed, a Master File for the HOLD file and a sequential data source called FOC\$HOLD are created. The data in FOC\$HOLD is then loaded into the HOLD file using an internally generated MODIFY procedure.

## Example

### Creating a HOLD File in FOCUS Format

The following example creates a subset of the CAR file:

```
TABLE FILE CAR
SUM SALES BY COUNTRY BY CAR BY MODEL
ON TABLE HOLD AS X1 FORMAT FOCUS
END
```

This request creates a single-segment FOCUS data source with a SEGTYPE of S3 (because it has three BY fields) named X1 FOCUS.

The X1 Master File is created by the request:

```
FILE=X1 ,SUFFIX=FOC
SEGNAME=SEG01, SEGTYPE=S03
FIELDNAME =COUNTRY ,E01 ,A10 , $
FIELDNAME =CAR ,E02 ,A16 , $
FIELDNAME =MODEL ,E03 ,A24 , $
FIELDNAME =SALES ,E05 ,I6 , $
```

Example      **Using PRINT to Create a FOCUS Data Source With a FOCLIST Field**

This example creates a single-segment FOCUS data source with a SEGTYPE of S4. Since this is a PRINT request, S4 stands for the addition of the three new BY fields and the FOCLIST value.

```
TABLE FILE CAR
PRINT SALES BY COUNTRY BY CAR BY MODEL
ON TABLE HOLD AS X2 FORMAT FOCUS INDEX MODEL
END
```

The Master File created by this request is:

```
FILE=X2 ,SUFFIX=FOC
SEGNAME=SEG01, SEGTYPE=S04
FIELDNAME =COUNTRY ,E01 ,A10 , $
FIELDNAME =CAR ,E02 ,A16 , $
FIELDNAME =MODEL ,E03 ,A24 ,FIELDTYPE=I, $
FIELDNAME =FOCLIST ,E02 ,I5 , $
FIELDNAME =SALES ,E05 ,I6 , $
```

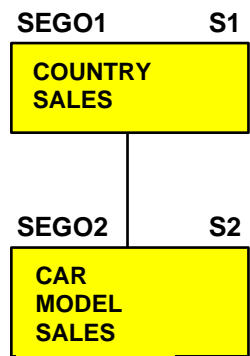
Example      **Creating a Two-Segment FOCUS Data Source**

The following request contains two SUM commands. The first, SUM SALES BY COUNTRY, creates a segment, with COUNTRY as the key and the summed values of SALES as a data field. The second, SUM SALES BY COUNTRY BY CAR BY MODEL, creates a descendant segment, with CAR and MODEL as the keys and SALES as a non-key field.

The COUNTRY field does not form part of the key to the second segment. COUNTRY is a key in the path to the second segment; any repetition of this value would be redundant.

```
TABLE FILE CAR
SUM SALES BY COUNTRY
SUM SALES BY COUNTRY BY CAR BY MODEL
ON TABLE HOLD AS X3 FORMAT FOCUS
END
```

A two-segment FOCUS data source with the following structure is created:



The Master File for this newly-created FOCUS data source is:

```
FILE=X3 ,SUFFIX=FOC
SEGNAME=SEG01, SEGTYPE=S01
FIELDNAME =COUNTRY ,E01 ,A10 ,,$
FIELDNAME =SALES ,E02 ,I6 ,,$
SEGNAME=SEG02, SEGTYPE=S02,PARENT=SEG01
FIELDNAME =CAR ,E03 ,A16 ,,$
FIELDNAME =MODEL ,E04 ,A24 ,,$
FIELDNAME =SALES ,E05 ,I6 ,,$
```

*Example*

**Creating a Three-Segment FOCUS Data Source**

In this example, each display command creates one segment.

The key to the root segment is the new BY field, COUNTRY, while the keys to the descendant segments are the new BY fields. The last segment uses the internal FOCLIST field as part of the key, since the display command is PRINT.

```
TABLE FILE CAR
SUM SALES BY COUNTRY BY CAR
SUM SALES BY COUNTRY BY CAR BY MODEL
PRINT SALES BY COUNTRY BY CAR BY MODEL BY BODY
ON TABLE HOLD AS X4 FORMAT FOCUS INDEX COUNTRY MODEL
END
```

The Master File is:

```
FILE=X4 ,SUFFIX=FOC
SEGNAME=SEG01, SEGTYPE =S02
FIELDNAME =COUNTRY ,E01 ,A10 ,FIELDTYPE=I,$
FIELDNAME =CAR ,E02 ,A16 ,,$
FIELDNAME =SALES ,E03 ,I6 ,,$
SEGNAME=SEG02, SEGTYPE =S01 ,PARENT=SEG01
FIELDNAME =MODEL ,E04 ,A24 ,FIELDTYPE=I,$
FIELDNAME =SALES ,E05 ,I6 ,,$
SEGNAME=SEG03, SEGTYPE =S02 ,PARENT=SEG02
FIELDNAME =BODYTYPE ,E06 ,A12 ,,$
FIELDNAME =FOCLIST ,E07 ,I5 ,,$
FIELDNAME =SALES ,E08 ,I6 ,,$
```

## Controlling Attributes in HOLD Master Files

The commands SET ASNAMES, SET HOLDLIST, and SET HOLDATTR enable you to control the FIELDNAME, TITLE, and ACCEPT attributes in HOLD Master Files. These commands are issued prior to the report request and remain in effect for the duration of the session, unless changed. For information about session duration in WebFOCUS environments, see the *Developing Applications* manual.

- The SET ASNAMES command causes text specified in an AS phrase to be used as the field name in the HOLD Master File, and to be concatenated to the beginning of the first field name specified in an ACROSS phrase. For details, *Controlling Field Names in a HOLD Master File* on page 11-14.
- The SET HOLDLIST command restricts fields in HOLD and PCHOLD files to those appearing in a request. That is, non-displaying fields in a request (those designated as NOPRINT fields) are not included in the HOLD file. For details, see *Controlling Field Names in the HOLD Master File* on page 11-16.
- The SET HOLDATTR command causes TITLE and ACCEPT attributes used in the original Master File to be used in the HOLD Master File. For details, see *Controlling TITLE and ACCEPT Attributes in a HOLD Master File* on page 11-20.

In addition, the SET HOLDSTAT command enables you to include comments and DBA information in the HOLD Master File. For more information about SET HOLDSTAT, see the *Describing Data* manual. For details about SET commands see the *Developing Applications* manual.

## Controlling Field Names in a HOLD Master File

When SET ASNAMES is set to ON or FOCUS, the literal specified in an AS phrase in a report request is used as the field name in a HOLD Master File. This command also controls how ACROSS fields are named in HOLD files.

### Syntax

#### How to Control Field Names in a HOLD Master File

`SET ASNAMES = [ON|OFF|FOCUS]`

where:

ON

Uses the literal specified in an AS phrase for the field name and controls the way ACROSS fields are named in HOLD files of any format.

OFF

Does not use the literal specified in an AS phrase as a field name in HOLD files, or affect the way ACROSS fields are named.

FOCUS

Uses the literal specified in an AS phrase as the field name and controls the way ACROSS fields are named only in HOLD files in FOCUS format. This is the default value.



## Reference

### Usage Notes for Controlling Field Names in HOLD Files

If no AS phrase is specified for a field, the field name from the original Master File is used. The TITLE attribute specified in the Master File will not be used unless SET HOLDATTR was previously issued.

To ensure that fields referenced more than once in a request have unique field names in the HOLD Master File, use SET ASNAMES.

- All characters are converted to uppercase.
- Special characters and blanks used in the AS phrase are preserved in the field name that is created when SET ASNAMES is used. When you refer to these non-standard field names in the newly created Master File, you must use single quotation marks around the field name.
- Text specified in an AS phrase that contains more than 66 characters is truncated to 66 characters in the Master File.
- Aliases are not carried over into the HOLD Master File. A new set of aliases is supplied automatically. These aliases are named E01 for the first field, E02 for the second, and so forth.
- Duplicate field names may occur in the newly created Master File as a result of truncation or the way AS phrases have been specified. In this case, refer to the fields by their aliases (E01, E02, and so forth).
- When commas are used as delimiters to break lines in the column heading, only the literal up to the first comma is used as the field name in the Master File. For example,  

```
PRINT COUNTRY AS 'PLACE,OF,ORIGIN'
```

produces the field name PLACE in the HOLD Master File.
- Unless SET ASNAMES=ON has been issued, field names exceeding 12 characters are used as field names in the HOLD Master File.
- When ACROSS is used in a report request and the results are extracted to a HOLD file, the columns generated by the ACROSS phrase all have the same field name in the HOLD Master File. If SET ASNAMES is issued, each new column may have a unique field name. This unique field name consists of the ASNAME value specified in the request's display command, concatenated to the beginning of the value of the field used in the ACROSS phrase. If several field names have the same letters, this approach will not work.

If an AS phrase is used for the fields in the ACROSS phrase, each new column will have a field name composed of the literal in the AS phrase concatenated to the beginning of the value of the first field used in the ACROSS phrase.

**Example**

**Controlling Field Names in the HOLD Master File**

In the following example, SET ASNAMES=ON causes the text in the AS phrase to be used as field names in the HOLD1 Master File. The two fields in the HOLD1 Master File, NATION and AUTOMOBILE, contain the data for COUNTRY and CAR.

```
SET ASNAMES=ON
TABLE FILE CAR
PRINT CAR AS 'AUTOMOBILE'
BY COUNTRY AS 'NATION'
ON TABLE HOLD AS HOLD1
END
```

The request produces the following Master File:

```
FILE=HOLD1 ,SUFFIX=FIX
SEGNAME=HOLD1, SEGTYPE=S01,$
FIELDNAME =NATION ,E01 ,A10 ,A12 , $
FIELDNAME =AUTOMOBILE ,E02 ,A16 ,A16 , $
```

**Example**

**Providing Unique Field Names With SET ASNAMES**

The following request generates a HOLD Master File with one unique field name for SALES and one for AVE.SALES. Both SALES and AVE.SALES would be named SALES, had SET ASNAMES not been used.

```
SET ASNAMES=ON
TABLE FILE CAR
SUM SALES AND AVE.SALES AS 'AVERAGESALES'
BY CAR
ON TABLE HOLD AS HOLD2
END
```

The request produces the following Master File:

```
FILE=HOLD2 ,SUFFIX=FIX
SEGNAME=HOLD2, SEGTYPE=S01,$
FIELDNAME =CAR ,E01 ,A16 ,A16 , $
FIELDNAME =SALES ,E02 ,I6 ,I04 , $
FIELDNAME =AVERAGESALES ,E03 ,I6 ,I04 , $
```

**Example**

**Using SET ASNAMES With the ACROSS Phrase**

The following request produces a HOLD Master File with the literal CASH concatenated to each value of COUNTRY:

```
SET ASNAMES=ON
TABLE FILE CAR
SUM SALES AS 'CASH'
ACROSS COUNTRY
ON TABLE HOLD AS HOLD3
END
```

The request produces the following Master File:

```
FILE=HOLD3 ,SUFFIX=FIX
SEGNAME=HOLD3, SEGTYPE=S01,$
FIELDNAME =CASHENGLAND ,E01 ,I6 ,I04 , $
FIELDNAME =CASHFRANCE ,E02 ,I6 ,I04 , $
FIELDNAME =CASHITALY ,E03 ,I6 ,I04 , $
FIELDNAME =CASHJAPAN ,E04 ,I6 ,I04 , $
FIELDNAME =CASHW GERMANY ,E05 ,I6 ,I04 , $
```

Without the SET ASNAMES command, every field in the HOLD FILE would be named COUNTRY.

To generate field names for ACROSS values that include only the field value, use the AS phrase followed by two single quotation marks as follows:

```
SET ASNAMES=ON
TABLE FILE CAR
SUM SALES AS ''
ACROSS COUNTRY
ON TABLE HOLD AS HOLD4
END
```

The resulting Master File looks like this:

```
FILE=HOLD4 ,SUFFIX=FIX
SEGNAME=HOLD4
FIELDNAME =ENGLAND ,E01 ,I6 ,I04 , $
FIELDNAME =FRANCE ,E02 ,I6 ,I04 , $
FIELDNAME =ITALY ,E03 ,I6 ,I04 , $
FIELDNAME =JAPAN ,E04 ,I6 ,I04 , $
FIELDNAME =W GERMANY ,E05 ,I6 ,I04 , $
```

# Controlling Fields in a HOLD Master File

You can use the SET HOLDLIST command to restrict fields in HOLD Master Files to those appearing in a request.

## Syntax

### How to Control Fields in a HOLD File

```
SET HOLDLIST = {PRINTONLY|ALL}
```

where:

PRINTONLY

Specifies that only those fields that would have appeared in the report are included in the generated HOLD file. Non-displaying fields in a request (that is, those designated as NOPRINT fields) are not included in the HOLD file.

ALL

Specifies that all display fields referenced in a request will appear in a HOLD file, including calculated values. ALL is the default value. OLD may be used as a synonym for ALL.

Note that SET HOLDLIST may also be issued from within a TABLE request. When used with MATCH, SET HOLDLIST always behaves as if HOLDLIST is set to ALL.

## Example

### Using HOLDLIST=ALL

When HOLDLIST is set to ALL, the following TABLE request produces a HOLD file containing all specified fields, including NOPRINT fields and values calculated with the COMPUTE command:

```
SET HOLDLIST=ALL

TABLE FILE CAR
PRINT CAR MODEL NOPRINT
COMPUTE TEMPSEATS=SEATS+1;
BY COUNTRY
ON TABLE HOLD
END

? HOLD
```

The result is:

| DEFINITION OF HOLD FILE: HOLD |       |        |
|-------------------------------|-------|--------|
| FIELDNAME                     | ALIAS | FORMAT |
| COUNTRY                       | E01   | A10    |
| CAR                           | E02   | A16    |
| MODEL                         | E03   | A24    |
| SEATS                         | E04   | I3     |
| TEMPSEATS                     | E05   | D12.2  |

## Example

### Using HOLDLIST= PRINTONLY

When HOLDLIST is set to PRINTONLY, the following report request produces a HOLD file containing only fields that would be displayed in report output:

```
SET HOLDLIST=PRINTONLY

TABLE FILE CAR
PRINT CAR MODEL NOPRINT
COMPUTE TEMPSEATS=SEATS+1;
BY COUNTRY
ON TABLE HOLD
END
```

? HOLD

The output is:

```
DEFINITION OF HOLD FILE: HOLD
FIELDNAME ALIAS FORMAT

COUNTRY E01 A10
CAR E02 A16
TEMPSEATS E03 D12.2
```

## Controlling the TITLE and ACCEPT Attributes in the HOLD Master File

The SET HOLDATTR command controls whether the TITLE and ACCEPT attributes in the original Master File are propagated to the HOLD Master File. SET HOLDATTR does not affect the way fields are named in the HOLD Master File.

Note that if a field in a data source does not have the TITLE attribute specified in the Master File, but there is an AS phrase specified for the field in a report request, the corresponding field in the HOLD file will be named according to the AS phrase.

## Syntax

### How to Control TITLE and ACCEPT Attributes

```
SET HOLDATTR =[ON|OFF|FOCUS]
```

where:

**ON**

Uses the TITLE attribute as specified in the original Master File in HOLD files in any format. The ACCEPT attribute will be propagated to the HOLD Master File only for HOLD files in FOCUS format.

**OFF**

Does not use the TITLE or ACCEPT attributes from the original Master File in the HOLD Master File.

**FOCUS**

Uses the TITLE and ACCEPT attributes only for HOLD files in FOCUS format. This is the default.

## **Example**

### **Controlling TITLE and ACCEPT Attributes in a HOLD Master File**

In this example, the Master File for the CAR data source specifies TITLE and ACCEPT attributes:

```
FILENAME=CAR2, SUFFIX=FOC
SEGNAME=ORIGIN, SEGTYPE=S1
 FIELDNAME =COUNTRY, COUNTRY, A10, TITLE='COUNTRY OF ORIGIN',
 ACCEPT='CANADA' OR 'ENGLAND' OR 'FRANCE' OR 'ITALY' OR
 'JAPAN' OR 'W GERMANY',
 FIELDTYPE=I,$
SEGNAME=COMP, SEGTYPE=S1, PARENT=ORIGIN
 FIELDNAME=CAR, CARS, A16, TITLE='NAME OF CAR',$
.
.
.
```

Using SET HOLDATTR=FOCUS, the following request

```
SET HOLDATTR = FOCUS
TABLE FILE CAR2
PRINT CAR
BY COUNTRY ON TABLE HOLD FORMAT FOCUS AS HOLD5
END
```

will produce this HOLD Master File:

```
FILE=HOLD5, SUFFIX=FOC
SEGNAME=SEG01, SEGTYPE=S02
FIELDNAME =COUNTRY ,E01 ,A10
 TITLE='COUNTRY OF ORIGIN',
 ACCEPT=CANADA ENGLAND FRANCE ITALY JAPAN 'W GERMANY',$
FIELDNAME =FOCLIST ,E02 ,I5 ,,$
FIELDNAME =CAR ,E03 ,A16 ,
 TITLE='NAME OF CAR' ,,$
```

# Keyed Retrieval From HOLD Files

Keyed retrieval is supported with any single segment SUFFIX=FIX data source or HOLD file that is sorted based on the key. Keyed retrieval can reduce the IOs incurred in reading extract files. The performance gains are accomplished by using the SEGTYPE parameter in the Master File to identify which fields comprise the logical key for sequential files.

- With FIXRETRIEVE=ON, the retrieval process stops when an equality or range test on the key holds true.
- With FIXRETRIEVE=OFF, all of the records from the sequential file are read and screening conditions are applied when creating the final report.

The ON TABLE HOLD command allows you to read one of the many supported data sources and create extract files. You can then join these fixed format sequential files to other data sources to narrow your view of the data. The concept of a logical key in a fixed format file permits qualified keyed searches for all records that match IF/WHERE tests for the first *n* KEY fields identified by the SEGTYPE attribute. Retrieval stops when the screening test detects values greater than those specified in the IF/WHERE test.

When a Master File is created for the extract file, a SEGTYPE of either *Sn* or *SHn* is added, based on the BY fields in the request. For example, PRINT *field* BY *field* creates a HOLD Master File with SEGTYPE=S1. Using BY HIGHEST *field* creates a Master with SEGTYPE=SH1.

## Syntax

### How to Control Keyed Retrieval for a HOLD File

```
SET FIXRET[RIEVE] = {ON|OFF}
```

where:

ON

Enables keyed retrieval. ON is the default setting.

OFF

Disables keyed retrieval.

### **Example**

### **Master File for Keyed Retrieval From a HOLD File**

The following Master File describes a fixed format sequential file with sorted values of MYKEY in ascending order from 1 to 100,000:

```
FILE=SORTED,SUFFIX=FIX,$
SEGNAME=ONE,SEGTYPE=S1,$
 FIELD=MYKEY,MK,I8,I8,$
 FIELD=MFIELD,MF,A10,A10,$
```

```
TABLE FILE SORTED
 PRINT MFIELD
 WHERE MYKEY EQ 100
END
```

In this instance, with FIXRETRIEVE=ON, retrieval stops when MYKEY reaches 101, vastly reducing the potential number of IOs, as only 101 records are read out of a possible 100,000 records.

### **Example**

### **Selection Criteria for Keyed Retrieval From an Extract File**

Selection criteria that include lists of equality values use keyed retrieval. For example,

```
{IF|WHERE} MYKEY EQ x OR y OR z
```

IF and WHERE tests can also include range tests. For example,

```
{IF|WHERE} MYKEY IS-FROM x TO y
```

The maximum number of vertical (BY) sort fields remains 32.

In using this feature, keep in mind that when adding unsorted records to a sorted HOLD file, out of sequence records will not be retrieved. For example, suppose that a sorted file contains the following three records

```
Key
1 1200
2 2340
3 4875
```

and you add the following record at the bottom of the file:

```
1 1620
```

With FIXRETRIEVE=ON, the new record with a key value of 1 would be omitted, as retrieval would stop as soon as a key value of 2 was encountered.



# Creating SAVE and SAVB Files

The SAVE command, by default, captures report output in ALPHA format as a simple sequential data source, without headings or subtotals. However, you can specify a variety of other formats for SAVE files, which are compatible with many software products. For example, you can specify SAVE formats to display report output in a Web page, a text document, a spreadsheet or word processing application, or to be used as input to other programming languages. For a list of supported formats see *Choosing Output File Formats* on page 11-26.

Regardless of format, the SAVE command does not create a Master File.

The SAVB command is a variation on the SAVE command. SAVB creates a data source without a Master File, but numeric fields are stored in BINARY format. You can use the SAVB file as input to a variety of applications. SAVB output is the same as the default output created by the HOLD command.

## Syntax

### How to Create a SAVE File

There are three ways to extract data to a SAVE file. In a report request, the syntax is

```
ON TABLE SAVE [AS filename] [FORMAT fmt] [MISSING {ON|OFF}]
```

or

```
save_field SAVE [AS filename] [FORMAT fmt] [MISSING {ON|OFF}]
```

At the command line or in a stored procedure, issue the SAVE command with appropriate options after the report is displayed.

The syntax is

```
SAVE [AS filename][FORMAT fmt][MISSING {ON|OFF}]
```

where:

*save\_field*

Is the name of the last field in the request.

*AS filename*

Specifies a name for the SAVE file. If you do not specify a file name, SAVE is used as the default name. Since each subsequent SAVE command overwrites the previous SAVE file, it is useful practice to code a distinct file name in each request to direct the extracted data to a separate file, thereby preventing it from being overwritten by the next SAVE command.

**FORMAT** *fmt*

Specifies the format of the SAVE file. The default format is ALPHA.

- To display as a Web page, choose:

*HTML*

This format is supported through the Web Interface.

- To use in a text document:

*ALPHA, PDF, WP, Text*

- To use in a spreadsheet application:

*DIF, EXCEL, LOTUS, SYLK*

For details about particular formats, see *Choosing Output File Formats* on page 11-26.

**MISSING**

Ensures that fields with the MISSING attribute set to ON will be carried over into the SAVE file. The default is MISSING OFF. For related information see Chapter 12, *Handling Records With Missing Field Values*.

**Example**

**Creating a SAVE File**

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
 BY DEPARTMENT
 ON TABLE SAVE
END
```

A description of the ALPHA (default SAVE format) file layout is displayed after the records are retrieved.

The output is:

|                             |       |        |        |
|-----------------------------|-------|--------|--------|
| NUMBER OF RECORDS IN TABLE= | 12    | LINES= | 12     |
|                             |       |        |        |
| EBCDIC RECORD NAMED SAVE    | ALIAS | FORMAT | LENGTH |
| FIELDNAME                   |       |        |        |
| DEPARTMENT                  | DPT   | A10    | 10     |
| LAST_NAME                   | LN    | A15    | 15     |
| FIRST_NAME                  | FN    | A10    | 10     |
| TOTAL                       |       |        | 35     |

## Syntax

### How to Create a SAVB File

There are three ways to create a SAVB file. In a TABLE request, the syntax is

```
ON TABLE SAVB [AS filename] [MISSING {ON|OFF}]
```

or

```
save_field SAVB [AS filename] [MISSING {ON|OFF}]
```

or

At the command line, issue SAVB with appropriate options after the report is displayed.

The syntax is

```
SAVB [AS filename] [MISSING {ON|OFF}]
```

where:

*save\_field*

Is the name of the last field in the request.

*AS filename*

Specifies a name for the SAVB file. If you do not specify a file name, SAVB is used as the default name. Since each subsequent SAVB command overwrites the previous SAVB file, it is useful practice to code a distinct file name in each request to direct the extracted data to a separate file, thereby preventing it from being overwritten by the next SAVB command.

**MISSING**

Ensures that fields with the MISSING attribute set to ON will be carried over into the SAVB file. The default is MISSING OFF. For related information, see Chapter 12, *Handling Records With Missing Field Values*.

**Example      Creating a SAVB File**

```
TABLE FILE SALES
PRINT PROD_CODE AND AREA
BY DATE
WHERE CITY IS 'STAMFORD' OR 'UNIONDALE'
ON TABLE SAVB
END
```

A description of the BINARY file is displayed after the records are retrieved.

The output is:

|                             |       |        |        |    |
|-----------------------------|-------|--------|--------|----|
| NUMBER OF RECORDS IN TABLE= |       | 10     | LINES= | 10 |
| INTERNAL RECORD NAMED SAVB  |       |        |        |    |
| FIELDNAME                   | ALIAS | FORMAT | LENGTH |    |
| DATE                        | DTE   | A4MD   | 4      |    |
| PROD_CODE                   | PCODE | A3     | 4      |    |
| AREA                        | LOC   | A1     | 4      |    |
| TOTAL                       |       |        | 12     |    |

# Choosing Output File Formats

You can select from a wide range of output formats to preserve your report output for use in any of the following ways:

- To display as or in a Web page, as a printed document, or in a text document.
- To process in another application, such as a spreadsheet, a database, a word processor, or a 3GL program.
- To send to another location, such as a browser or PC.
- To extract a subset of the original data source in order to generate multi-step reports.

For details on each of the supported formats, including the commands that support them (HOLD, PCHOLD, SAVE) and the operating environments in which they are available, see the corresponding reference topics.

**Reference****FORMAT ALPHA****Description:**

Saves report output as fixed-format character data. When created as a HOLD file, a corresponding Master File is also created.

ALPHA is the default SAVE format.

Text fields are supported in ALPHA-formatted files. See *Rules for Text Fields in Output Files* on page 11-40.

**Use:**

For display in a text document; for further reporting in FOCUS or WebFOCUS; as a transaction file for modifying a data source.

**Supported with the commands:**

HOLD; PCHOLD; SAVE

The PCHOLD variation transfers the data and the Master File from a Web server to a browser.

**Available in:**

WebFOCUS; FOCUS for S/390.

**Reference****FORMAT BINARY****Description:**

Saves report data and stores numeric fields as binary numbers; also creates a Master File.

BINARY is the default format for HOLD files.

**Use:**

For further reporting in FOCUS or WebFOCUS; as transaction file for modifying a data source.

**Supported with the commands:**

HOLD; PCHOLD

The PCHOLD variation transfers the data from a Web server to a browser.

**Available in:**

WebFOCUS; FOCUS for S/390

## **Reference**

### **FORMAT COMMA**

**Description:**

Saves the data values as a variable-length text file with fields separated by commas and with character values enclosed in double quotation marks. All blanks within fields are retained. This format is the industry standard comma-delimited format.

This format also includes a built-in safety feature, which allows embedded quotes within text fields. This feature inserts a second double quote (") adjacent to the existing one. For example, if you input Joe "Smitty" Smith, the output will be Joe ""Smitty"" Smith.

The extension or file type for this format is PRN. This format type does not create a Master File.

**Note:**

- Smart date fields and dates formatted as I or P fields with date format options are treated as numeric and are not enclosed in double quotation marks in the output file. Dates formatted as alphanumeric fields with date format options are treated as alphanumeric and enclosed in double quotation marks.
- Continental decimal notation (CDN=ON|SPACE|QUOTE) is not supported. A comma within a number would be interpreted as two separate columns by a destination application such as Microsoft Access.

**Use:**

For further processing in a database application. This format type can be imported into applications such as Excel or Lotus.

**Supported with the commands:**

HOLD, SAVE

**Available in:**

WebFOCUS; FOCUS for S/390

## Reference

### FORMAT COM

**Description:**

Saves the data values as a variable-length text file with fields separated by commas and with character values enclosed in double quotation marks. Leading blanks are removed from numeric fields and trailing blanks are removed from character fields. To issue a request against this data source, the setting PCOMMA=ON is required.

This format also includes a built-in safety feature, which allows embedded quotes within text fields. This feature inserts a second double quote (") adjacent to the existing one. For example, if you input Joe "Smitty" Smith, the output will be Joe ""Smitty"" Smith.

The extension or file type for this format is CSV. A Master File is created for this format type when the command used to create the output file is HOLD. The SUFFIX in the generated Master File is COM.

**Note:**

- Smart date fields and dates formatted as I or P fields with date format options are treated as numeric and are not enclosed in double quotation marks in the output file. Dates formatted as alphanumeric fields with date format options are treated as alphanumeric and enclosed in double quotation marks.
- Continental decimal notation (CDN=ON|SPACE|QUOTE) is not supported. A comma within a number would be interpreted as two separate columns by a destination application such as Microsoft Access.

**Use:**

For further processing in a database application. This format type can be imported into applications such as Excel or Lotus.

**Supported with the commands:**

HOLD, SAVE

**Available in:**

FOCUS for S/390

## **Reference**

### **FORMAT COMT**

#### **Description:**

Saves the column headings in the first row of the output file. It produces a variable-length text file with fields separated by commas and with character values enclosed in double quotation marks. Leading blanks are removed from numeric fields and trailing blanks are removed from character fields. This format is required by certain software packages such as Microsoft Access.

This format also includes a built-in safety feature, which allows embedded quotes within text fields. This feature inserts a second double quote (") adjacent to the existing one. For example, if you input Joe "Smitty" Smith, the output will be Joe ""Smitty"" Smith.

The extension or file type for this format is CSV. A Master File is created for this format type when the command used to create the output file is HOLD. The SUFFIX in the generated Master File is COMT.

#### **Note:**

- Smart date fields and dates formatted as I or P fields with date format options are treated as numeric and are not enclosed in double quotation marks in the output file. Dates formatted as alphanumeric fields with date format options are treated as alphanumeric and enclosed in double quotation marks.
- Continental decimal notation (CDN=ON|SPACE|QUOTE) is not supported. A comma within a number would be interpreted as two separate columns by a destination application such as Microsoft Access.

#### **Use:**

For further processing in a database application. This format type can be imported into applications such as Excel or Lotus.

#### **Supported with the commands:**

HOLD, SAVE

#### **Available in:**

FOCUS for S/390

## **Reference**

### **FORMAT DB2**

#### **Description:**

Creates a DB2 table, if you have the DB2 Data Adapter and permission to create tables.

#### **Use:**

For further processing in a database application.

#### **Supported with the command:**

HOLD

#### **Available in:**

WebFOCUS; FOCUS for S/390.



## Reference

### FORMAT DIF

**Description:**

Saves report data and field headings and creates a character file that can be easily imported into most spreadsheet packages.

**Note:** Microsoft Excel SR-1 is no longer supported for HOLD FORMAT DIF. To open these reports, use either Microsoft Excel SR-2 or Microsoft Excel 2000.

**Use:**

For display or processing in a spreadsheet application.

**Supported with the commands:**

HOLD, PCHOLD, SAVE

The PCHOLD variation transfers the data from a Web server to a browser.

**Available in:**

WebFOCUS; FOCUS for S/390.

## Reference

### FORMAT EXCEL

**Description:**

Captures report output as a Microsoft Excel spreadsheet file, including data and column titles, but without report headings, footings, subheadings, or subfootings. If the report request contains an ACROSS phrase and specifies FORMAT EXCEL, column titles are not included in the output.

Text fields are not supported with HOLD FORMAT EXCEL.

**Note:** Microsoft Excel SR-1 is no longer supported for HOLD FORMAT EXCEL. To open these reports, use either Microsoft Excel SR-2 or Microsoft Excel 2000.

**Use:**

For display or processing in a spreadsheet application.

**Supported with the commands:**

HOLD, PCHOLD, SAVE

**Available in:**

WebFOCUS; FOCUS for S/390.

In FOCUS for S/390, the recommended transfer mechanism is FTP in binary mode. In CMS, the file type of the resulting file is XLS. On a PC, the extension should be .xls.

## **Reference**

### **FORMAT FOCUS**

**Description:**

Creates a FOCUS data source. Four files result: a HOLD data file, a HOLD Master File, and two work files. For detailed information see *Holding Report Output in FOCUS Format* on page 11-9.

Text fields are supported for FOCUS output files. See *Rules for Text Fields in Output Files* on page 11-40.

**Use:**

For further processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390.

## **Reference**

### **FORMAT FUSION**

**Description:**

Captures the report data and creates a Fusion multi-dimensional data source.

You must either specify the name of an Access File that supplies the name of the Fusion data source or supply the name of the data source in the HOLD command as follows:

```
[ACCESS FILE filename|DATASET fully-qualified_data_source_name]
[INDEX field 1..., fieldn]
```

Requires the parameter setting MASTER=NEW.

**Use:**

For further processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390.

## **Reference**

### **FORMAT HTML**

**Description:**

Creates a complete HTML document that can be viewed in a Web browser.

The PCHOLD variation transfers the data from a Web server to a browser.

**Use:**

For display as a Web page.

**Supported with the commands:**

HOLD, PCHOLD, SAVE

**Available in:**

WebFOCUS; FOCUS for S/390 as a feature of the Web Interface for FOCUS.

**Reference****FORMAT HTMTABLE****Description:**

Creates an output file that contains only an HTML table. The output produced is not a complete HTML document, however, the file can be included in another HTML document using the Dialogue Manager command -HTMLFORM. For details see the documentation on Dialogue Manager in the *Developing Applications* manual.

**Note:** When issuing HOLD AS name FORMAT HTMTABLE, you must specify a different name than the .html file name used in the -HTMLFORM name.

**Use:**

For embedding reports and graphs in an existing HTML document.

**Supported with the commands:**

HOLD, PCHOLD, SAVE

The PCHOLD variation also transfers the data from a Web server to a browser.

**Available in:**

WebFOCUS; FOCUS for S/390 as a feature of the Web Interface for FOCUS.

**Reference****FORMAT INGRES****Description:**

Creates an Ingres table, if you have the Ingres Data Adapter and permission to create tables.

**Use:**

For further processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390 when used as a client to iWay.

**Reference****FORMAT INTERNAL****Description:**

Saves report output without padding the values of integer and packed fields. For details see *Saving Report Output in INTERNAL Format* on page 11-42.

**Use:**

For accurate processing by 3GL programs.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390.

**Reference**

**FORMAT LOTUS**

**Description:**

Captures all the columns of the report in a character file that LOTUS 1-2-3 can then import. All alphanumeric fields are enclosed in quotation marks. Columns are separated by commas.

**Use:**

For display and processing in a spreadsheet application.

**Supported with the commands:**

HOLD, PCHOLD, SAVE (WebFOCUS browser version only)

**Available in:**

WebFOCUS; FOCUS for S/390.

In VM/CMS, the LOTUS file has a file name of PRN and allocates a scratch data set to the file HOLD.

**Reference**

**FORMAT PDF**

**Description:**

Saves the report output in Adobe's Portable Document Format, which allows precise placement of output (that is, all formatting options such as headings, footings, and titles) correctly aligned on the physical page, so the report looks exactly as it would when printed.

**Use:**

For display as a printed document.

**Supported with the commands:**

HOLD, PCHOLD, SAVE (WebFOCUS only)

**Available in:**

WebFOCUS; FOCUS for S/390 as a feature of the Web Interface for FOCUS.

**Reference**

**FORMAT POSTSCRIPT (PS)**

**Description:**

Creates an output file in PostScript format, which supports headings, footings, and totals.

PS is an abbreviation for POSTSCRIPT. In CMS, the file type is PS.

PS also supports: Outline fonts, ISO Latin font encoding, and EPS image format.

**Use:**

For display as a printed document.

**Supported with the command:**

HOLD, PCHOLD

**Available in:**

WebFOCUS (browser version); FOCUS for S/390.

*Reference*

**FORMAT REDBRICK**

**Description:**

Creates a Red Brick table, if you have the Redbrick Data Adapter and permission to create tables.

**Use:**

For further processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390 when used as a client to iWay.

*Reference*

**FORMAT SQL**

**Description:**

Creates an SQL/DS table, if you have the SQL/DS Data Adapter and permission to create tables.

**Use:**

For processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390.

*Reference*

**FORMAT SQLDBC**

**Description:**

Creates a Teradata table, if you have the Teradata Data Adapter and permission to create tables.

**Use:**

For processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390.

**Reference**

**FORMAT SQLINF**

**Description:**

Creates an Informix table, if you have the Informix Data Adapter and permission to create tables.

**Use:**

For processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390 when used as a client to iWay.

**Reference**

**FORMAT SQLMSS**

**Description:**

Creates a Microsoft SQL Server table, if you have the Microsoft SQL Data Adapter and permission to create tables.

**Use:**

For processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390 when used as a client to iWay.

**Reference**

**FORMAT SQLODBC**

**Description:**

Creates an SQLODBC table if you have the current ODBC Data Adapter and permission to create tables.

**Use:**

For processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390 when used as a client to iWay.

*Reference*

**FORMAT SQLORA**

**Description:**

Creates an Oracle table, if you have the Oracle Data Adapter and permission to create tables.

**Use:**

For processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390.

*Reference*

**FORMAT SQLSYB**

**Description:**

Creates a Sybase table, if you have the Sybase Data Adapter and permission to create tables.

**Use:**

For processing in a database application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390 when used as a client to iWay.

*Reference*

**FORMAT SYLK**

**Description:**

Captures all the columns of the report request in a character file for Microsoft's spreadsheet program Multiplan. The generated file cannot have more than 9,999 rows.

**Use:**

For display and processing in a spreadsheet application.

**Supported with the command:**

HOLD

**Available in:**

WebFOCUS; FOCUS for S/390.

## Reference

### FORMAT TABT

**Description:**

Creates an output file in tab-delimited format that includes column headings in the first row. The TABT format includes a built-in safety feature, which allows embedded quotes within text fields. This feature inserts a second double quote (") adjacent to the existing one. For example, if you input Joe "Smitty" Smith, the output will be Joe ""Smitty"" Smith. The TABT format also includes the following features:

- The first row contains field names.
- All trailing blanks are stripped from alpha[An] fields.
- All leading blanks are stripped from numeric [/Dx.y, /Fx.y, /Px.y, and /In] fields.
- There is a 32K record length limit in the output file.
- A Master File is created when the command used to create the output file is HOLD. The Master File behaves exactly as in FORMAT ALPHA, except for the inclusion of double quotes.

**Note:** Blank field names display as blank column titles. This may result in an error when attempting to use the file as input to various applications.

**Use:**

For importing data to Windows-based applications such as MS Access and Excel.

**Supported with the command:**

HOLD, SAVE

**Available in:**

WebFOCUS; FOCUS for S/390

## Reference

### FORMAT WP

**Description:**

Captures the entire report output—including headings, footings, and subtotals—and creates a text file that can be easily incorporated into most word processing packages.

Text fields are supported in WP format. See *Rules for Text Fields in Output Files* on page 11-40.

To control whether a carriage control character is included in column 1 of each page of the report output use the following syntax:

```
[ON TABLE] HOLD AS filename FORMAT WP [CC|NOCC]
```

NOCC excludes carriage control characters. The position reserved for those characters remains in the file, but is blank. CC includes carriage control characters and, in MVS, creates the HOLD file with RECFM VBA. To include page control information in the WP file, you can also specify the TABPAGENO option in a heading or the SET PAGE=OFF command. The character 1 in the column 1 indicates the start of a new page.



The following rules summarize FORMAT WP carriage control options:

- The CC option always inserts the carriage control character.
- The NOCC option always omits the carriage control character.
- When you issue HOLD FORMAT WP without the CC or NOCC option:  
SET PAGE NUM=OFF and SET PAGE NUM=TOP always insert the carriage control character.  
SET PAGE NUM=NOPAGE always omits the carriage control character.  
SET PAGE NUM=ON inserts the carriage control character if TABPAGENO is included in the heading and omits the carriage control character if TABPAGENO is not included in the heading.

**Tip:**

HOLD FORMAT WP does not change the number of lines per page. If you wish to change the number of lines per page, issue one or a combination of the commands SET PRINT=OFFLINE, SET SCREEN=PAPER, or SET SCREEN=OFF.

In MVS, the WP file is created with a record format of VB when the carriage control character is omitted and with a record format of VBA when the carriage control character is inserted.

The maximum output for HOLD FORMAT WP is 248 characters.

FORMAT WP retains headings, footings, and subtotals.

**Use:**

For display in a text document

**Supported with the commands:**

HOLD, PCHOLD, SAVE

**Available in:**

WebFOCUS; FOCUS for S/390.

## **Reference**

### **Rules for Text Fields in Output Files**

Text fields can be used in HOLD and SAVE files that have the following formats: ALPHA, WP, and FOCUS.

- You can include as many text fields in the file as needed. However, you must specify text fields after non-text fields in the display list (PRINT..., SUM..., and so forth).
- You can specify only one text field in the display list, and no non-text fields, in a request that includes an ACROSS phrase.

The following rules apply to missing data for text fields in HOLD and SAVE files:

- A blank line is valid data. An end-of-text mark will indicate the end of the field.
- If there is no text for a field, a single period (.) followed by blanks will appear in the HOLD or SAVE file.
- If MISSING=ON during data extraction, a period (.) is written out to the HOLD or SAVE file.
- If MISSING=OFF during data extraction, a blank is written out to the HOLD or SAVE file.

For details see Chapter 12, *Handling Records With Missing Field Values*.

In environments that support FIXFORM, due to limitations in the use of text fields with FIXFORM, the following restrictions apply:

- When you use the command FIXFORM FROM HOLD, the HOLD file may not contain more than one text field and the text field must be the last field in the Master File.

When HOLD files are read using FIXFORM, the interpretation of missing text depends on whether the field's designation is MISSING=ON in the Master File, or conditional (C) in the FIXFORM format description, or some combination of the two.

## Example

### Applying Text Field Rules in HOLD Files

The following request extracts data in the HOLD file CRSEHOLD:

```
TABLE FILE COURSES
PRINT COURSE_CODE DESCRIPTION
ON TABLE HOLD AS CRSEHOLD
END

TABLE FILE CRSEHOLD
PRINT *
END
```

The output is:

```
101 This course provides the DP professional with the skills
needed to create, maintain, and report from FOCUS databases.
%$
200 Anyone responsible for designing FOCUS databases will benefit
from this course, which provides the skills needed to design large,
complex databases and tune existing ones.
%$
201 This is a course in FOCUS efficiencies.
%$
```

The first record of the HOLD file contains data for COURSE\_CODE 101, followed by the DESCRIPTION field. The data for this text field extends into the next record, beginning at Column 1, and continues to the end of the HOLD record. It is immediately followed by the end-of-text mark (%\$) on a line by itself. The next record contains new data for the next COURSE\_CODE and DESCRIPTION.

If the report uses two text fields, the first record will contain data for the first text field. After the end-of-text mark is written, the next text field will be displayed. This formatting applies to all file formats except WP, in which the report is saved exactly as it appears on the screen.

## Saving Report Output in INTERNAL Format

HOLD files pad binary integer and packed decimal data values to a full word boundary. For example, a three-digit integer field (I3), is stored as four bytes in a HOLD file. In order for third generation programs, such as COBOL, to be able to read HOLD files in an exact manner, you may need to save the fields in the HOLD file without any padding.

To suppress field padding in the HOLD file, do the following in the report request:

- Specify formats for the integer and packed fields that should *not* be padded (in order to override the ACTUAL formats used for the corresponding USAGE formats specified in the Master File).
- Specify HOLD FORMAT INTERNAL for the report output.

### Syntax

### How to Suppress Field Padding in HOLD Files

```
SET HOLDLIST = PRINTONLY
TABLE FILE filename
display_command fieldname/[In|Pn.d]
.
.
ON TABLE HOLD AS name FORMAT INTERNAL
END
```

where:

**PRINTONLY**

Causes your report request to propagate the HOLD file with only the specified fields displaying in the report output. If you do not issue this setting, an extra field containing the padded field length is included in the HOLD file. For details, see *Controlling Fields in a HOLD Master File* on page 11-18.

**fieldname/[In|Pn.d]**

Specifies formats for integer and packed fields for which you wish to suppress padding. These formats will override the ACTUAL formats used for the display formats in the Master File. For details, see *Usage Notes for Suppressing Padded Fields in HOLD Files* on page 11-43.

Note that floating point double-precision (D), floating pointing point single-precision (F), and Alphanumeric (A) fields are not affected by HOLD FORMAT INTERNAL.

**FORMAT INTERNAL**

Saves the HOLD file without padding for specified integer and packed decimal fields.

For an illustration, see *Creating a HOLD File With HOLD FORMAT INTERNAL* on page 11-1.

# Reference

## Usage Notes for Suppressing Padded Fields in HOLD Files

- Integer fields (I) of one, two, three, or four bytes produce four byte integers without HOLD FORMAT INTERNAL.
- For packed decimal fields ( $Pn.d$ ),  $n$  is the total number of digits and  $d$  is the number of digits to the right of the decimal point. The number of bytes is derived by dividing  $n$  by 2 and adding 1.

The syntax is

`bytes = INT (n/2) + 1`

where:

`INT (n/2)`

Is the greatest integer after dividing by 2.

- HOLD FORMAT INTERNAL does not affect floating point double precision (D) and floating point single precision (F) fields. D remains at eight bytes and F at four bytes.
- Alphanumeric fields automatically inherit their length from their source Master File, and are not padded to a full word boundary. Therefore, they are also not affected by HOLD FORMAT INTERNAL.
- If a format override is not large enough to contain the data values, the values are truncated. Truncation may cause the data in the HOLD file to be incorrect in the case of an integer. For packed data and integers, truncation occurs for the high order digits so the remaining low order digits will resemble the digits from the correct values.

To avoid incorrect results, be sure that the format you specify is large enough to contain the data values or the values will be incorrect.

# Example

## Creating a HOLD File Without HOLD FORMAT INTERNAL

In this example, the values of ACTUAL for RETAIL\_COST, DEALER\_COST, and SEATS are all padded to a full word in binary.

```
TABLE FILE CAR
PRINT CAR COUNTRY RETAIL_COST DEALER_COST SEATS
ON TABLE HOLD AS DJG
END
```

?DJG

The request creates and displays the following Master File:

```
FILE=DJG ,SUFFIX=FIX
SEGNAME=DJG ,SEGTYPE=S0
FIELDNAME =CAR ,E01 ,A16 ,A16 , $
FIELDNAME =COUNTRY ,E02 ,A10 ,A12 , $
FIELDNAME =RETAIL_COST ,E03 ,D7 ,D08 , $
FIELDNAME =DEALER_COST ,E04 ,D7 ,D08 , $
FIELDNAME =SEATS ,E05 ,I3 ,I04 , $
```

**Example**                      **Creating a HOLD File With HOLD FORMAT INTERNAL**

In this example, DEALER\_COST and RETAIL\_COST are defined in the Master File as D fields, but the request overrides RETAIL\_COST as an I2 field and DEALER\_COST as a P3 field.

```
SET HOLDLIST=PRINTONLY
TABLE FILE CAR
PRINT CAR COUNTRY RETAIL_COST/I2 DEALER_COST/P3 SEATS/I1
ON TABLE HOLD AS HINT3 FORMAT INTERNAL
END

?HINT3
```

This creates and displays the following Master File:

```
FILE=HINT3 ,SUFFIX=FIX
SEGNAME=HINT3 ,SEGTYPE=S0
FIELDNAME =CAR ,E01 ,A16 ,A16 , $
FIELDNAME =COUNTRY ,E02 ,A10 ,A10 , $
FIELDNAME =RETAIL_COST,E03 ,I6 ,I02 , $
FIELDNAME =DEALER_COST,E04 ,P4 ,P02 , $
FIELDNAME =SEATS ,E05 ,I4 ,I01 , $
```

The values of ACTUAL for the overridden fields are I2, P2, and I1. DEALER\_COST has an ACTUAL of P2 because the format override, P3, means 3 display digits that can be stored in 2 actual digits.

---

## CHAPTER 12

# Handling Records With Missing Field Values

### Topics:

- Irrelevant Report Data
- Missing Field Values
- Handling a Missing Segment Instance
- Setting the NODATA Character String

Missing data is defined as data that is missing from a report because it is not relevant or because it does not exist in the data source. Report output that involves averaging and counting calculations or the display of parent segment instances may be affected by missing data. Data can be missing from reports and calculations for the following reasons:

- Data is not relevant to a particular row and column in a report. See *Irrelevant Report Data* on page 12-2.
- A field in a segment instance does not have a data value. See *Missing Field Values* on page 12-3.
- A parent segment instance does not have child instances (missing segment instances). See *Handling a Missing Segment Instance* on page 12-13.

**Note:** To run the examples in this topic, you must run the stored procedures EMPMISS and SALEMISS to add missing data to the EMPLOYEE and SALES data sources, respectively.

# Irrelevant Report Data

Data can be missing from a report row or column because it is not relevant. The missing or inapplicable value is indicated by the NODATA default character, a period (.).

**Tip:**  
You may specify a more meaningful NODATA value by issuing the SET NODATA command (see *Setting the NODATA Character String* on page 12-20).

## Example

### Irrelevant Report Data

The following request shows how the default NODATA character displays missing data in a report.

```
TABLE FILE EMPLOYEE
PRINT SALARY
BY LAST_NAME
BY FIRST_NAME
ACROSS DEPARTMENT
END
```

The output is:

|           |            | DEPARTMENT  |             |
|-----------|------------|-------------|-------------|
|           |            | MIS         | PRODUCTION  |
| LAST_NAME | FIRST_NAME |             |             |
| BANNING   | JOHN       | .           | \$29,700.00 |
| BLACKWOOD | ROSEMARIE  | \$21,780.00 | .           |
| CROSS     | BARBARA    | \$27,062.00 | .           |
| GREENSPAN | MARY       | \$9,000.00  | .           |
| IRVING    | JOAN       | .           | \$26,862.00 |
| JONES     | DIANE      | \$18,480.00 | .           |
| MCCOY     | JOHN       | \$18,480.00 | .           |
| MCKNIGHT  | ROGER      | .           | \$16,100.00 |
| ROMANS    | ANTHONY    | .           | \$21,120.00 |
| SMITH     | MARY       | \$13,200.00 | .           |
|           | RICHARD    | .           | \$9,500.00  |
| STEVENS   | ALFRED     | .           | \$11,000.00 |

The salary for an employee working in the production department displays in the PRODUCTION column; the salary for an employee working in the MIS department displays in that column. The corresponding value in the PRODUCTION or MIS column, respectively, is missing because the salary displays only under the department where the person is employed.



# Missing Field Values

Missing values within segment instances occur when the instances exist, but some fields in the instances lack values.

When fields in instances lack values, numeric fields are assigned the value of 0, and alphanumeric fields the value of blank. These default values appear in reports and are used in all calculations performed by the SUM and COUNT display commands, DEFINE commands, and prefix operators such as MAX. and AVE.

To prevent the use of these default values in calculations (which might then give erroneous results), you can add the MISSING attribute to the field declaration in the Master File, for either a real or a virtual field. When the MISSING attribute is set to ON the missing values are marked with a special internal code to distinguish them from blanks or zeros, and the missing values are ignored in calculations. In reports, the internal code is represented by the SET NODATA value—a period (.), by default. See *Setting the NODATA Character String* on page 12-20.

For example, missing data for a field in a segment instance may occur when the data values are unknown, as in the following scenario. Suppose that the employees recorded in the EMPLOYEE data source are due for a pay raise by a certain date, but the amount of the raise has not been determined. The company enters the date for each employee into the data source without the salary amounts; the salaries will be entered later. Each date is an individual instance in the salary history segment, but the new salary for each date instance is missing. Suppose further that a report request averages the SALARY field (SUM AVE.SALARY). The accuracy of the resulting average depends on whether the missing values for the SALARY field are treated as zeros (MISSING=OFF) or as internal codes (MISSING=ON).

## Example

### Counting With Missing Values

Suppose the CURR\_SAL field appears in 12 segment instances. In three of those instances, the field was given no value. Nevertheless, the display command

```
COUNT CURR_SAL
```

counts 12 occurrences of the CURR\_SAL field. This occurs because the MISSING attribute is OFF by default, so the missing values are included in the count. If you wanted to exclude the missing data from the count, you could set MISSING ON.

## Example

### Averaging With Missing Values

Suppose you had the following records of data for a field:

```
.
.
1
3
```

The numeric values in the first two records are missing (indicated by the periods); the last two records have values of 1 and 3. If you average these fields without the MISSING attribute (MISSING OFF), a value of 0 is supplied for the two records that are missing values. Thus, the average of the records is  $(0+0+1+3)/4$ , or 1. If you use the MISSING ON attribute, the two missing values are ignored, calculating the average as  $(1+3)/2$ , or 2.

## MISSING Attribute in the Master File

In some applications, the default values (blanks and zeros) may represent valid data rather than the absence of information. However, if this is not the case, you can include the MISSING attribute after the field format in the Master File declaration for the field with the missing values. The MISSING attribute can be used with an actual field in the data source or a virtual field that you are defining in the Master File.

For example, the following field declaration specifies the MISSING attribute for the RETURNS field:

```
FIELDNAME=RETURNS, ALIAS=RTN, FORMAT=I4, MISSING=ON, $
```

The next declaration specifies the MISSING attribute for a virtual field called PROFIT:

```
DEFINE PROFIT/D7 MISSING ON NEEDS SOME DATA = RETAIL_COST - DEALER_COST; $
```

To ensure that missing values are handled properly for virtual fields, you can set the MISSING attribute ON for the virtual field in the DEFINE command, and specify whether you want to apply the calculation if some or all values are missing. For related information on the SOME and ALL phrases, see *How to Specify Missing Values in a DEFINE Command* page 12-6.

When the MISSING attribute is set to ON in a field declaration, the field containing no data is marked with a special internal code, rather than with blanks or zeros. During report generation, the SUM and COUNT commands and all prefix operators (for example, AVE., MAX., MIN.) will exclude the missing data in their computations. For related information about the MISSING attribute and field declarations, see the *Describing Data* manual.

#### Note:

- You may add MISSING field attributes to the Master File at any time. However, MISSING attributes affect only data entered into the data source after the attributes were added.
- Key fields are needed to identify a record; therefore, key fields should not be identified as missing.

Example

Handling Missing Values With the MISSING Attribute

This example illustrates the difference between a field with MISSING ON and one without. In it a virtual field, X>Returns, without the MISSING attribute, is set to equal a real field, RETURNS, with the MISSING attribute declared in the Master File. When the field with the MISSING attribute (RETURNS) is missing a value, the corresponding value of X>Returns is 0, since a data source field that is missing a value is evaluated as 0 (or blank) for the purpose of computation (see MISSING Attribute in a DEFINE Command on page 12-5).

The following request defines the virtual field:

```
DEFINE FILE SALES
X>Returns/I4 = RETURNS;
END
```

Now issue the following report request:

```
TABLE FILE SALES
SUM CNT.X>Returns CNT.RETURNS AVE.X>Returns AVE.RETURNS
END
```

Remember that the field X>Returns has the same value as RETURNS except when RETURNS is missing a value, in which case the X>Returns value is 0.

The output is:

|                    |                  |                  |                |
|--------------------|------------------|------------------|----------------|
| PAGE               |                  | 1                |                |
| X_RETURNS<br>COUNT | RETURNS<br>COUNT | AVE<br>X_RETURNS | AVE<br>RETURNS |
| -----              | -----            | -----            | -----          |
| 22                 | 20               | 2                | 3              |

The count for the RETURNS field is lower than the count for X>Returns and the average for RETURNS is higher than for X>Returns because the missing values in RETURNS are *not* part of the calculations.

For an illustration in which the MISSING attribute is set for a virtual field see *Handling Missing Values for Virtual Fields With SOME and ALL* on page 12-8.

MISSING Attribute in a DEFINE Command

You can set the MISSING attribute ON in a DEFINE command to enable a virtual field with missing values to be interpreted and represented correctly in reports.

A DEFINE expression, used to derive the values of the virtual field, can contain real fields that have missing values. However, when used to derive the value of a virtual field, a data source field that is missing a value is evaluated as 0 or blank for the purpose of computation, even if the MISSING attribute has been set to ON for that field in the Master File.

To ensure that missing values are handled properly for virtual fields, you can set the MISSING attribute ON for the virtual field in the DEFINE command, and specify whether you want to apply the calculation if some or all values are missing. See *How to Specify Missing Values in a DEFINE Command* on page 12-6.

Note that you cannot use the MISSING attribute in a COMPUTE command in a report request.

## Syntax

### How to Specify Missing Values in a DEFINE Command

To control how missing values are handled for a virtual field, use the following syntax

```
field[/format] MISSING {ON|OFF} [NEEDS] {SOME|ALL} [DATA] = expression;
```

where:

*field*

Is the name of the virtual field created by the DEFINE command.

*/format*

Is the format of the virtual field. The default is D12.2.

MISSING

ON enables the value of the virtual field to be interpreted as missing (that is, distinguished by the special internal code from an intentionally entered zero or blank), and represented by the NODATA character in reports.

OFF treats missing values for numeric fields as zeros, and missing values for alphanumeric fields as blanks. This is the default value.

NEEDS

Is optional. It helps to clarify the meaning of the command.

SOME

Indicates that if at least one field in the expression has a value, the virtual field has a value (the field's missing values are evaluated as 0 or blank in the calculation). If all of the fields in the expression are missing values, the virtual field is missing its value. This value is the default.

ALL

Indicates that if all the fields in the expression have values, the virtual field has a value. If at least one field in the expression has a missing value, the virtual field also has a missing value.

DATA

Is optional. It helps to clarify the meaning of the command.

*expression*

Is a valid expression from which the virtual field derives its value.

Example

Handling Missing Values for a Virtual Field With MISSING OFF

The following request illustrates the use of two fields, RETURNS and DAMAGED, to define the NO\_SALE field. Both the RETURNS and DAMAGED fields have the MISSING attribute set to ON in the SALES Master File, yet whenever one of these fields is missing a value, that field is evaluated as 0.

```
DEFINE FILE SALES
NO_SALE/I4 = RETURNS + DAMAGED;
END

TABLE FILE SALES
PRINT RETURNS AND DAMAGED AND NO_SALE
BY CITY BY DATE BY PROD_CODE
END
```

The output is:

|           |       |           |         |         |         |
|-----------|-------|-----------|---------|---------|---------|
| PAGE      | 1     |           |         |         |         |
|           |       |           |         |         |         |
| CITY      | DATE  | PROD_CODE | RETURNS | DAMAGED | NO_SALE |
| NEW YORK  | 10/17 | B10       | 2       | 3       | 5       |
|           |       | B17       | 2       | 1       | 3       |
|           |       | B20       | 0       | 1       | 1       |
|           |       | C13       | .       | 6       | 6       |
|           |       | C14       | 4       | .       | 4       |
|           |       | C17       | 0       | 0       | 0       |
|           |       | D12       | 3       | 2       | 5       |
|           |       | E1        | 4       | 7       | 11      |
|           |       | E2        | .       | .       | 0       |
|           |       | E3        | 4       | 2       | 6       |
| NEWARK    | 10/18 | B10       | 1       | 1       | 2       |
|           | 10/19 | B12       | 1       | 0       | 1       |
| STAMFORD  | 12/12 | B10       | 10      | 6       | 16      |
|           |       | B12       | 3       | 3       | 6       |
|           |       | B17       | 2       | 1       | 3       |
|           |       | C13       | 3       | 0       | 3       |
|           |       | C7        | 5       | 4       | 9       |
|           |       | D12       | 0       | 0       | 0       |
|           |       | E2        | 9       | 4       | 13      |
|           |       | E3        | 8       | 9       | 17      |
| UNIONDALE | 10/18 | B20       | 1       | 1       | 2       |
|           |       | C7        | 0       | 0       | 0       |

Notice that the products C13, C14, and E2 in the New York section all show missing values for either RETURNS or DAMAGED because the MISSING ON attribute has been set in the Master File. However, the calculation that determines the value of NO\_SALE interprets these missing values as zeros because MISSING ON has not been set for the virtual field.

Example Handling Missing Values for Virtual Fields With SOME and ALL

The following request illustrates how to use the DEFINE command with the MISSING attribute to specify that if either some or all of the field values referenced in a DEFINE command are missing, the virtual field should also be missing its value.

The SOMEDATA field contains a value if either the RETURNS or DAMAGED field contains a value. Otherwise, SOMEDATA is missing its value. The ALLDATA field contains a value only if both the RETURNS and DAMAGED fields contain values. Otherwise, ALLDATA is missing its value.

```
DEFINE FILE SALES
SOMEDATA/I5 MISSING ON NEEDS SOME=RETURNS + DAMAGED;
ALLDATA/I5 MISSING ON NEEDS ALL=RETURNS + DAMAGED;
END
```

```
TABLE FILE SALES
PRINT RETURNS AND DAMAGED SOMEDATA ALLDATA
BY CITY BY DATE BY PROD_CODE
END
```

The output is:

PAGE 1

| CITY      | DATE  | PROD_CODE | RETURNS | DAMAGED | SOMEDATA | ALLDATA |
|-----------|-------|-----------|---------|---------|----------|---------|
| NEW YORK  | 10/17 | B10       | 2       | 3       | 5        | 5       |
|           |       | B17       | 2       | 1       | 3        | 3       |
|           |       | B20       | 0       | 1       | 1        | 1       |
|           |       | C13       | .       | 6       | 6        | .       |
|           |       | C14       | 4       | .       | 4        | .       |
|           |       | C17       | 0       | 0       | 0        | 0       |
|           |       | D12       | 3       | 2       | 5        | 5       |
|           |       | E1        | 4       | 7       | 11       | 11      |
|           |       | E2        | .       | .       | .        | .       |
| NEWARK    | 10/18 | E3        | 4       | 2       | 6        | 6       |
|           |       | B10       | 1       | 1       | 2        | 2       |
|           |       | B12       | 1       | 0       | 1        | 1       |
| STAMFORD  | 12/12 | B10       | 10      | 6       | 16       | 16      |
|           |       | B12       | 3       | 3       | 6        | 6       |
|           |       | B17       | 2       | 1       | 3        | 3       |
|           |       | C13       | 3       | 0       | 3        | 3       |
|           |       | C7        | 5       | 4       | 9        | 9       |
|           |       | D12       | 0       | 0       | 0        | 0       |
|           |       | E2        | 9       | 4       | 13       | 13      |
|           |       | E3        | 8       | 9       | 17       | 17      |
| UNIONDALE | 10/18 | B20       | 1       | 1       | 2        | 2       |
|           |       | C7        | 0       | 0       | 0        | 0       |

## Testing for a Segment With a Missing Field Value

You can specify WHERE criteria to identify segment instances with missing field values. You cannot use these tests to identify missing instances. However, you can use the ALL PASS parameter to test for missing instances. See *Handling a Missing Segment Instance* on page 12-13.

### Syntax

### How to Test for a Segment With a Missing Field Value

To test for a segment with missing field values, the syntax is:

```
WHERE field {IS|EQ} MISSING
```

To test for the presence of field values, the syntax is:

```
WHERE field {NE|IS-NOT} MISSING
```

A WHERE criterion that tests a numeric field for 0 or an alphanumeric field for blanks also retrieves instances for which the field has a missing value.

### Example

### Testing for a Missing Field Value

The following request illustrates the use of MISSING to display grocery items (by code) for which the number of packages returned by customers is missing.

```
TABLE FILE SALES
PRINT RETURNS
BY CITY BY DATE BY PROD_CODE
WHERE RETURNS IS MISSING
END
```

The output is:

PAGE 1

| CITY     | DATE  | PROD_CODE | RETURNS |
|----------|-------|-----------|---------|
| ----     | ----  | -----     | -----   |
| NEW YORK | 10/17 | C13       | .       |
|          |       | E2        | .       |

**Example**

**Testing for an Existing Field Value**

The following request illustrates the use of MISSING to display only those grocery items for which the number of packages returned by customers is not missing.

```
TABLE FILE SALES
PRINT RETURNS
BY CITY BY DATE BY PROD_CODE
WHERE RETURNS IS-NOT MISSING
END
```

The output is:

PAGE 1

| CITY      | DATE  | PROD_CODE | RETURNS |
|-----------|-------|-----------|---------|
| NEW YORK  | 10/17 | B10       | 2       |
|           |       | B17       | 2       |
|           |       | B20       | 0       |
|           |       | C14       | 4       |
|           |       | C17       | 0       |
|           |       | D12       | 3       |
|           |       | E1        | 4       |
|           |       | E3        | 4       |
| NEWARK    | 10/18 | B10       | 1       |
|           | 10/19 | B12       | 1       |
| STAMFORD  | 12/12 | B10       | 10      |
|           |       | B12       | 3       |
|           |       | B17       | 2       |
|           |       | C13       | 3       |
|           |       | C7        | 5       |
|           |       | D12       | 0       |
|           |       | E2        | 9       |
|           |       | E3        | 8       |
| UNIONDALE | 10/18 | B20       | 1       |
|           |       | C7        | 0       |



Example      Testing for a Blank or Zero

The following request displays grocery items that either were never returned or for which the number of returned packages was never recorded:

```
TABLE FILE SALES
PRINT RETURNS
BY CITY BY DATE BY PROD_CODE
WHERE RETURNS EQ 0
END
```

The output is:

PAGE      1

| CITY      | DATE  | PROD_CODE | RETURNS |
|-----------|-------|-----------|---------|
| <hr/>     |       |           |         |
| NEW YORK  | 10/17 | B20       | 0       |
|           |       | C13       | .       |
|           |       | C17       | 0       |
|           |       | E2        | .       |
| STAMFORD  | 12/12 | D12       | 0       |
| UNIONDALE | 10/18 | C7        | 0       |

Example      Excluding Missing Values From a Test

To display only those items that have not been returned by customers, you need two WHERE criteria: one to restrict the number of returns to 0, the other to exclude missing values, as in the following request:

```
TABLE FILE SALES
PRINT RETURNS
BY CITY BY DATE BY PROD_CODE
WHERE RETURNS EQ 0
WHERE RETURNS IS-NOT MISSING
END
```

The output is:

PAGE      1

| CITY      | DATE  | PROD_CODE | RETURNS |
|-----------|-------|-----------|---------|
| <hr/>     |       |           |         |
| NEW YORK  | 10/17 | B20       | 0       |
|           |       | C17       | 0       |
| STAMFORD  | 12/12 | D12       | 0       |
| UNIONDALE | 10/18 | C7        | 0       |

## Preserving Missing Data Values in an Output File

The ability to distinguish between missing data and default values (blanks and zeros) in fields can be carried over into output files. If the retrieved and processed information displayed the NODATA string in a report, the NODATA string can be stored in the output file. For related information, see Chapter 11, *Saving and Reusing Report Output*.

### Syntax

#### How to Distinguish Missing Data in an Extract File

```
ON TABLE {HOLD|SAVE|SAVB} MISSING {ON|OFF}
```

where:

##### HOLD

Creates an extract file that can be used for subsequent reports. The default value for MISSING is ON.

##### SAVE

Creates a text extract file that can be used by other programs. The default for MISSING is OFF.

##### SAVB

Creates a binary extract file that can be used by other programs. The default for MISSING is OFF.

HOLD files can be created with both the MISSING and FORMAT ALPHA options, specified in any order. For example:

```
ON TABLE HOLD FORMAT ALPHA MISSING OFF
```

```
ON TABLE HOLD MISSING OFF FORMAT ALPHA
```

### Example

#### Incorporating MISSING Values in an Extract File

The HOLD, SAVE, and SAVB commands can incorporate the MISSING attribute with display commands. For example:

```
TABLE FILE SALES
SUM RETURNS AND SAVE MISSING ON
BY CITY BY DATE BY PROD_CODE
END
```

# Handling a Missing Segment Instance

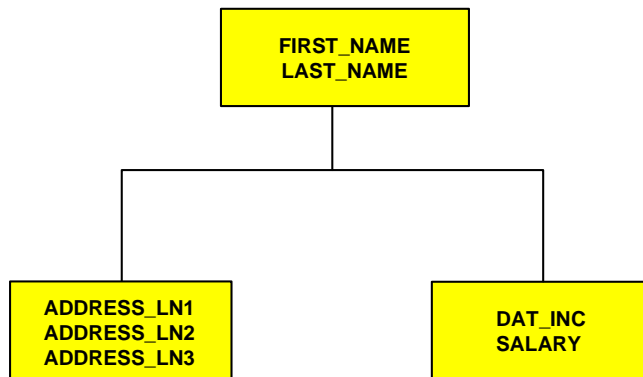
In multi-segment data sources, when an instance in a parent segment does not have descendant instances, the nonexistent descendant instances are called missing instances.

When you write a request from a data source that has missing segment instances, the missing instances affect the report. For example, if the request names fields in a segment and its descendants, the report omits parent segment instances that have no descendants. It makes no difference whether fields are display fields or sort fields.

When an instance is missing descendants in a child segment, the instance, its parent, the parent of its parent, and so on up to the root segment, is called a short path. Unique segments are never considered to be missing.

For example, consider the following subset of the EMPLOYEE data source (see Appendix A, *Master Files and Diagrams*).

- The top segment contains employee names.
- The left segment contains addresses.
- The right segment contains each employee's salary history: the date the employee was granted a new salary and the amount of the salary.



Suppose some employees are paid by an outside agency. None of these employees have a company salary history. Instances referring to these employees in the salary history segment are missing.

Nonexistent descendant instances affect whether parent segment instances are included in report results. The SET ALL command and the ALL. prefix enable you to include parent segment data in reports.

For illustrations of how missing segment instances impact reporting, see *Reporting Against Segments Without Descendant Instances* on page 12-14 and *Reporting Against Segments With Descendant Instances* on page 12-15.

Example Reporting Against Segments Without Descendant Instances

The following request displays the salary histories for each employee:

```
TABLE FILE EMPLOYEE
PRINT SALARY
BY LAST_NAME BY FIRST_NAME
BY DAT_INC
END
```

However, two employees, Davis and Gardner, are omitted from the following report because the LAST\_NAME and FIRST\_NAME fields belong to the root segment, and the DAT\_INC and SALARY fields belong to the descendant salary history segment. Since Davis and Gardner have no descendant instances in the salary history segment, they are omitted from the report.

The output is:

|           |            |          |             |
|-----------|------------|----------|-------------|
| PAGE      | 1          |          |             |
| LAST_NAME | FIRST_NAME | DAT_INC  | SALARY      |
| BANNING   | JOHN       | 82/08/01 | \$29,700.00 |
| BLACKWOOD | ROSEMARIE  | 82/04/01 | \$21,780.00 |
| CROSS     | BARBARA    | 81/11/02 | \$25,775.00 |
|           |            | 82/04/09 | \$27,062.00 |
| GREENSPAN | MARY       | 82/04/01 | \$8,650.00  |
|           |            | 82/06/11 | \$9,000.00  |
| IRVING    | JOAN       | 82/01/04 | \$24,420.00 |
|           |            | 82/05/14 | \$26,862.00 |
| JONES     | DIANE      | 82/05/01 | \$17,750.00 |
|           |            | 82/06/01 | \$18,480.00 |
| MCCOY     | JOHN       | 82/01/01 | \$18,480.00 |
| MCKNIGHT  | ROGER      | 82/02/02 | \$15,000.00 |
|           |            | 82/05/14 | \$16,100.00 |
| ROMANS    | ANTHONY    | 82/07/01 | \$21,120.00 |
| SMITH     | MARY       | 82/01/01 | \$13,200.00 |
|           | RICHARD    | 82/01/04 | \$9,050.00  |
|           |            | 82/05/14 | \$9,500.00  |
| STEVENS   | ALFRED     | 81/01/01 | \$10,000.00 |
|           |            | 82/01/01 | \$11,000.00 |

**Example****Reporting Against Segments With Descendant Instances**

The following request displays the average salary and second address line of each employee. The data source contains Davis' address, but not Gardner's.

```
TABLE FILE EMPLOYEE
SUM AVE.SALARY
AND ADDRESS_LN2
BY LAST_NAME BY FIRST_NAME
END
```

This report displays Davis' name even though Davis has no salary history, because Davis has an instance in the descendant address segment. The report omits Gardner entirely, because Gardner has neither a salary history nor an address.

The output is:

PAGE 1

| LAST_NAME | FIRST_NAME | AVE<br>SALARY | ADDRESS_LN2         |
|-----------|------------|---------------|---------------------|
| -----     | -----      | -----         | -----               |
| BANNING   | JOHN       | \$29,700.00   | APT 4C              |
| BLACKWOOD | ROSEMARIE  | \$21,780.00   | 3704 FARRAGUT RD.   |
| CROSS     | BARBARA    | \$26,418.50   | 147-15 NORTHERN BLD |
| DAVIS     | ELIZABETH  | .             | 2530 AMSTERDAM AVE. |
| GREENSPAN | MARY       | \$8,825.00    | 13 LINDEN AVE.      |
| IRVING    | JOAN       | \$25,641.00   | 123 E 32 ST.        |
| JONES     | DIANE      | \$18,115.00   | 235 MURRAY HIL PKWY |
| MCCOY     | JOHN       | \$18,480.00   | 2 PENN PLAZA        |
| MCKNIGHT  | ROGER      | \$15,550.00   | 117 HARRISON AVE.   |
| ROMANS    | ANTHONY    | \$21,120.00   | 271 PRESIDENT ST.   |
| SMITH     | MARY       | \$13,200.00   | 2 PENN PLAZA        |
|           | RICHARD    | \$9,275.00    | 136 E 161 ST.       |
| STEVENS   | ALFRED     | \$10,500.00   | 2 PENN PLAZA        |

## Including Missing Instances in Reports With the ALL. Prefix

If a request excludes parent segment instances that lack descendants, you can include the parent instances by attaching the ALL. prefix to one of the fields in the parent segment.

Note that if the request contains WHERE or IF criteria that screen fields in segments that have missing instances, the report omits parent instances even when you use the ALL. prefix. To include the instances, use the SET ALL=PASS command described in *Including Missing Instances in Reports With the SET ALL Command* on page 12-17.

### Example

### Including Missing Segment Instances With the ALL. Prefix

The following request displays the salary history of each employee. Although employees Elizabeth Davis and David Gardner have no salary histories, they are included in the report.

```
TABLE FILE EMPLOYEE
PRINT SALARY
BY ALL.LAST_NAME BY FIRST_NAME
BY DAT_INC
END
```

The output is:

PAGE 1

| LAST_NAME | FIRST_NAME | DAT_INC  | SALARY      |
|-----------|------------|----------|-------------|
| BANNING   | JOHN       | 82/08/01 | \$29,700.00 |
| BLACKWOOD | ROSEMARIE  | 82/04/01 | \$21,780.00 |
| CROSS     | BARBARA    | 81/11/02 | \$25,775.00 |
|           |            | 82/04/09 | \$27,062.00 |
| DAVIS     | ELIZABETH  | .        | .           |
| GARDNER   | DAVID      | .        | .           |
| GREENSPAN | MARY       | 82/04/01 | \$8,650.00  |
|           |            | 82/06/11 | \$9,000.00  |
| IRVING    | JOAN       | 82/01/04 | \$24,420.00 |
|           |            | 82/05/14 | \$26,862.00 |
| JONES     | DIANE      | 82/05/01 | \$17,750.00 |
|           |            | 82/06/01 | \$18,480.00 |
| MCCOY     | JOHN       | 82/01/01 | \$18,480.00 |
| MCKNIGHT  | ROGER      | 82/02/02 | \$15,000.00 |
|           |            | 82/05/14 | \$16,100.00 |
| ROMANS    | ANTHONY    | 82/07/01 | \$21,120.00 |
| SMITH     | MARY       | 82/01/01 | \$13,200.00 |
|           | RICHARD    | 82/01/04 | \$9,050.00  |
|           |            | 82/05/14 | \$9,500.00  |
| STEVENS   | ALFRED     | 81/01/01 | \$10,000.00 |
|           |            | 82/01/01 | \$11,000.00 |

## Including Missing Instances in Reports With the SET ALL Command

You can include parent instances with missing descendants by issuing the SET ALL command before executing the request.

**Note:** A request with WHERE or IF criteria, which screen fields in a segment that has missing instances, omits instances in the parent segment even if you use the SET ALL=ON command. To include these instances in the report, use the SET ALL=PASS command.

### Syntax

### How to Include a Parent Instance With Missing Descendants

`SET ALL= {OFF|ON|PASS}`

where:

#### [OFF](#)

Omits parent instances that are missing descendants from the report. OFF is the default value.

#### [ON](#)

Includes parent instances that are missing descendants in the report. However, a test on a missing segment fails, and causes the parent to be omitted from the output. It is comparable to the ALL. prefix.

#### [PASS](#)

Includes parent instances that are missing descendants, even if WHERE or IF criteria exist to screen fields in the descendant segments that are missing instances (that is, a test on a missing segment passes).

## Example

### Including Missing Segment Instances With SET ALL

Consider this request which includes employees without salary histories because the ALL=PASS command is set.

If the ALL=ON command had been used, the employees without salary histories would have been omitted because of the WHERE criteria.

```
SET ALL=PASS
```

```
TABLE FILE EMPLOYEE
PRINT SALARY
BY LAST_NAME BY FIRST_NAME
BY DAT_INC
WHERE SALARY LT 20000
END
```

The output is:

PAGE 1

| LAST_NAME | FIRST_NAME | DAT_INC  | SALARY      |
|-----------|------------|----------|-------------|
| DAVIS     | ELIZABETH  | .        | .           |
| GARDNER   | DAVID      | .        | .           |
| GREENSPAN | MARY       | 82/04/01 | \$8,650.00  |
|           |            | 82/06/11 | \$9,000.00  |
| JONES     | DIANE      | 82/05/01 | \$17,750.00 |
|           |            | 82/06/01 | \$18,480.00 |
| MCCOY     | JOHN       | 82/01/01 | \$18,480.00 |
| MCKNIGHT  | ROGER      | 82/02/02 | \$15,000.00 |
|           |            | 82/05/14 | \$16,100.00 |
| SMITH     | MARY       | 82/01/01 | \$13,200.00 |
|           | RICHARD    | 82/01/04 | \$9,050.00  |
|           |            | 82/05/14 | \$9,500.00  |
| STEVENS   | ALFRED     | 81/01/01 | \$10,000.00 |
|           |            | 82/01/01 | \$11,000.00 |



## Testing for Missing Instances in FOCUS Data Sources

You can use the ALL PASS parameter to produce reports that include only parent instances with missing descendant values. To do so, write the request to screen out all existing instances in the segment with missing instances. After you set the ALL parameter to PASS, the report will display only the parent instances missing descendants.

### *Example*

#### Testing for a MISSING Instance

The following request screens all salary instances since no SALARY can be both equal to 0 and not equal to 0.

```
SET ALL = PASS
```

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME
WHERE SALARY EQ 0
WHERE SALARY NE 0
END
```

The output is:

PAGE 1

| LAST_NAME | FIRST_NAME |
|-----------|------------|
| -----     | -----      |
| DAVIS     | ELIZABETH  |
| GARDNER   | DAVID      |

# Setting the NODATA Character String

In a report, the NODATA character string indicates no data or inapplicable data. The default NODATA character is a period, however, you can change this character designation.

Syntax

How to Set the NODATA String

```
SET NODATA = string
```

where:

```
string
```

Is the character string used to indicate missing data in reports. The default string is a period (.). The string may be a maximum of 11 characters. Common choices are NONE, N/A, and MISSING.

Example

Setting NODATA Not to Display Characters

If you do not want any characters, issue the command:

```
SET NODATA = ' '
```

Example

Setting the NODATA Character String

In the following request, the NODATA character string is set to MISSING. The word MISSING displays on the report instead of the default period.

```
SET NODATA=MISSING
```

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY LAST_NAME BY FIRST_NAME
ACROSS DEPARTMENT
END
```

The output is:

PAGE1

| LAST_NAME | FIRST_NAME | DEPARTMENT  |             |
|-----------|------------|-------------|-------------|
|           |            | MIS         | PRODUCTION  |
| BANNING   | JOHN       | MISSING     | \$29,700.00 |
| BLACKWOOD | ROSEMARIE  | \$21,780.00 | MISSING     |
| CROSS     | BARBARA    | \$27,062.00 | MISSING     |
| DAVIS     | ELIZABETH  | \$.00       | MISSING     |
| GARDNER   | DAVID      | MISSING     | \$.00       |
| GREENSPAN | MARY       | \$9,000.00  | MISSING     |
| IRVING    | JOAN       | MISSING     | \$26,862.00 |
| JONES     | DIANE      | \$18,480.00 | MISSING     |
| MCCOY     | JOHN       | \$18,480.00 | MISSING     |
| MCKNIGHT  | ROGER      | MISSING     | \$16,100.00 |
| ROMANS    | ANTHONY    | MISSING     | \$21,120.00 |
| SMITH     | MARY       | \$13,200.00 | MISSING     |
|           | RICHARD    | MISSING     | \$9,500.00  |
| STEVENS   | ALFRED     | MISSING     | \$11,000.00 |

---

## CHAPTER 13

# Joining Data Sources

### Topics:

- Types of Joins
- How the JOIN Command Works
- Creating an Equijoin
- Using a Conditional Join
- Preserving Virtual Fields During Join Parsing
- Displaying Joined Structures
- Clearing Joined Structures

You can join two or more related data sources to create a larger integrated data structure from which you can report in a single request. The joined structure is virtual—it is a way of accessing multiple data sources as if they were a single data source. Up to 16 joins can be in effect at one time, for a total of 64 segments, depending on the number of active segments and the number and length of the fields (there is a 32K limit on the length of all fields).

With the exception of comma-delimited, tab-delimited, or token-delimited files, you can join any supported data source type. For details see *Data Sources You Can and Cannot Join* on page 13-3.

## Types of Joins

You can join data sources using one of two join types. The first type of join is known as an equijoin. The second type of join is a conditional join. When deciding which of the two types of joins to use, it is important to know that when there is an equality condition between two data sources, it is more efficient to use an equijoin rather than a conditional join.

You can use an equijoin structure when you are joining two (or more) data sources that have two fields, one in each data source, with formats (character, numeric, or date) and values in common. Joining a product code field in a sales data source (the host file) to the product code field in a product data source (the cross-referenced file) is an example of an equijoin. For more information on using equijoins, see *Creating an Equijoin* on page 13-13.

The second type of join, the conditional join, uses WHERE-based syntax to specify joins based on WHERE criteria, not just on equality between fields. Additionally, the host and cross-referenced join fields do not have to contain matching formats. Suppose you have a data source that lists employees by their ID number (the host file) and another data source that lists training courses and the employees who attended those courses (the cross-referenced file). Using a conditional join, you could join employee ID in the host file to employee ID in the cross-referenced file to determine which employees took training courses in a given date range (the WHERE condition). For more information on conditional joins, see *Using a Conditional Join* on page 13-23.

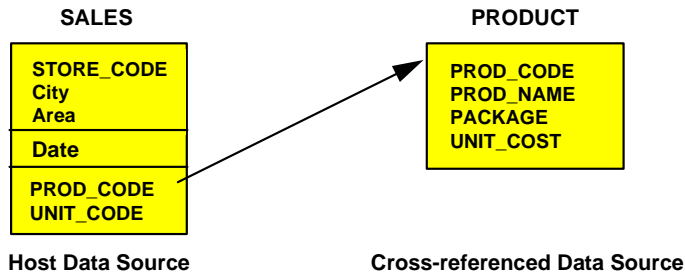
Both types of joins allow you to perform unique and non-unique joins. A unique, or one-to-one, join structure matches one value in the host data source to one value in the cross-referenced data source. Joining an employee ID in an employee data source to an employee ID in a salary data source is an example of a unique equijoin structure.

A non-unique, or one-to-many, join structure matches one value in the host data source to multiple values in the cross-referenced field. Joining employee ID in a company's employee data source to employee ID in a data source that lists all the training classes offered by that company would result in a listing of all courses taken by each employee, or a joining of the one instance of each ID in the host file to the multiple instances of that ID in the cross-referenced file.

For more information on unique and non-unique joins, see *Unique and Non-Unique Joined Structures* on page 13-4.

**Example****Joined Data Structure**

Consider the SALES and PRODUCT data sources. Each store record in SALES may contain many instances of the PROD\_CODE field. It would be redundant to store the associated product information with each instance of the product code; instead, PROD\_CODE in the SALES data source is joined to PROD\_CODE in the PRODUCT data source. PRODUCT contains a single instance of each product code and related product information, saving space and making maintenance of product information easier. The joined structure, which is an example of an equijoin, is illustrated below:

**Reference****Data Sources You Can and Cannot Join**

The use of data sources as host files and cross-referenced files in joined structures depends on the types of data sources you are joining:

- Typically, joins can be established between any FOCUS-readable files.
- You cannot join comma-delimited, tab-delimited, or token-delimited files.
- Data sources protected by DBA security may be joined, with certain restrictions. For details, see *Notes on DBA Security for Joined Data Structures* on page 13-4.
- Conditional joins are supported only for FOCUS, VSAM, ADABAS, IMS, and all relational data sources.

## Reference

### Notes on DBA Security for Joined Data Structures

- You can join a data source with DBA protection to another data source with DBA protection, as long as they use the same DBA password.
- In addition, you can join DBA protected data sources with different passwords by adding the DBAFILE attribute to your security definition. The DBAFILE attribute names a central Master File that contains different passwords and restrictions for several Master Files. If you use a DBAFILE, a user can set separate passwords for each file using the syntax:

```
SET PASS = pswd1 IN file1, pswd2 IN file2
```

Individual DBA information remains in effect for each file in the JOIN. For details about the DBAFILE attribute, see the *Describing Data* manual.

- You can also join a DBA-protected host file to an unprotected cross-referenced file. The DBA information is taken from the host file.

## Unique and Non-Unique Joined Structures

In a unique joined structure, one value in the host field corresponds to one value in the cross-referenced field. In a non-unique joined structure, one value in the host field corresponds to multiple values in the cross-referenced field.

The ALL parameter in a JOIN command indicates that the joined structure is non-unique.

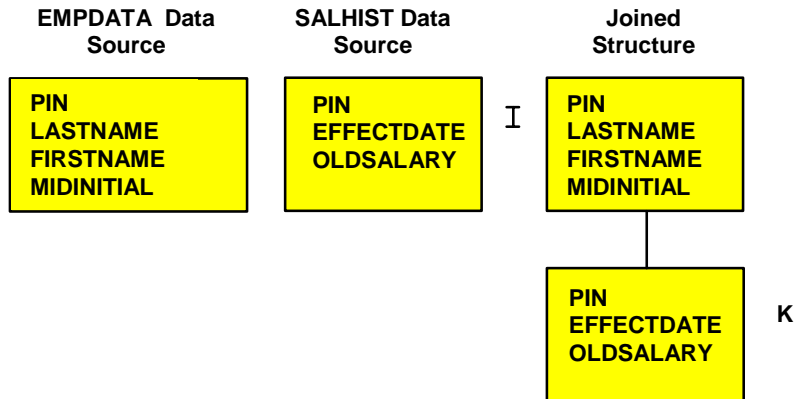
- Omit the ALL parameter only when you are sure that the joined structure is unique. Omitting the ALL parameter reduces overhead.
- The ALL parameter will not interfere with the proper creation of the joined structure even if it is unique. Use the ALL parameter if you are not sure whether the joined structure is unique. This ensures that your reports will contain all relevant data from the cross-referenced file, regardless of whether the structure is unique.

## Example

### A Unique Equijoin Structure

The following example illustrates a unique joined structure. Two FOCUS data sources are joined together: an EMPDATA data source and a SALHIST data source. Both data sources are organized by pin, and they are joined on a PIN field in the root segments of both files. Each PIN has one segment instance in the EMPDATA data source and one instance in the SALHIST data source. To join these two data sources, issue this JOIN command:

```
JOIN PIN IN EMPDATA TO PIN IN SALHIST
```



Example      A Non-Unique Equijoin Structure

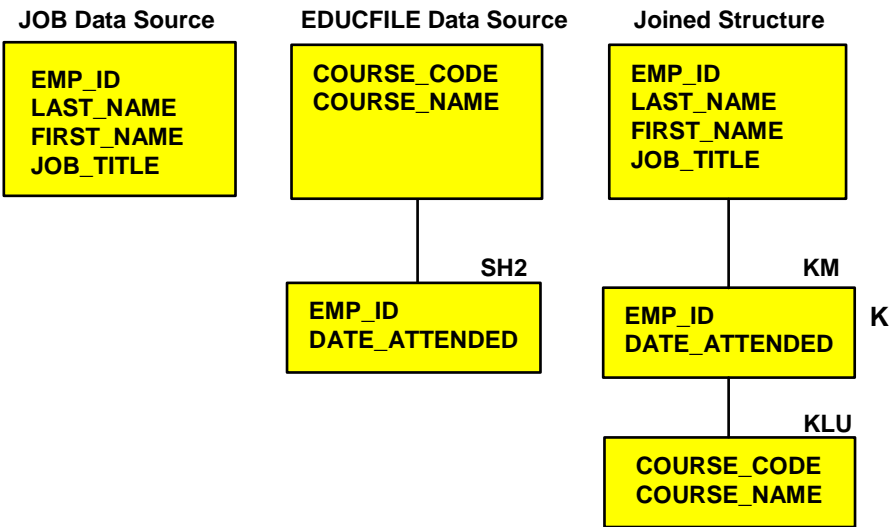
If a field value in the host file can appear in many segment instances in the cross-referenced file, then you should include the ALL phrase in the JOIN syntax. This structure is called a non-unique joined structure.

For example, assume that two FOCUS data sources are joined together: the JOB data source and an EDUCFILE data source which records employee attendance at in-house courses. The joined structure is shown in the following diagram.

The JOB data source is organized by employee, but the EDUCFILE data source is organized by course. The data sources are joined on the EMP\_ID field. Since an employee has one position but can attend several courses, the employee has one segment instance in the JOB data source but can have many instances in the EDUCFILE data source, as many instances as courses attended.

To join these two data sources, issue the following JOIN command, using the ALL phrase:

```
JOIN EMP_ID IN JOB TO ALL EMP_ID IN EDUCFILE
```





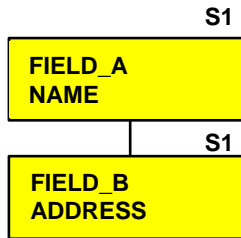
## Recursive Joined Structures

You can join a FOCUS or IMS data source to itself, creating a recursive structure. In the most common type of recursive structure, a parent segment is joined to a descendant segment, so that the parent becomes the child of the descendant. This technique (useful for storing bills of materials, for example) enables you to report from data sources as if they have more segment levels than they really do.

### Example

#### Understanding Recursive Joined Structures

For example, the GENERIC data source shown below consists of Segments A and B.

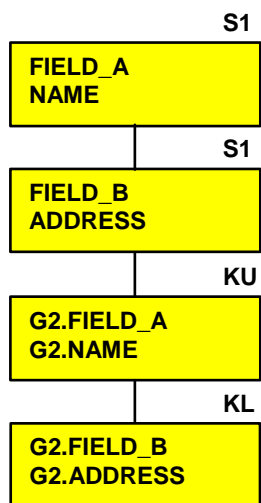


#### GENERIC Data Source

The following request creates a recursive structure:

```
JOIN FIELD_B IN GENERIC TAG G1 TO FIELD_A IN GENERIC TAG G2 AS RECURSIV
```

This results in the joined structure (shown below).



**GENERIC Data Source  
Recursively Joined**

Note that the two segments are repeated on the bottom. To refer to the fields in the repeated segments (other than the field to which you are joining), prefix the tag names to the field names and aliases and separate them with a period or append the first four characters of the JOIN name to the field names and aliases. In the above example, the JOIN name is RECURSIV. You should refer to FIELD\_B in the bottom segment as G2.FIELD\_B (or RECUFIELD\_B). For related information, see *Usage Notes for Recursive Joined Structures* on page 13-9.

## Reference

### Usage Notes for Recursive Joined Structures

- You must either specify a unique JOIN name or use tag names in the JOIN command. Otherwise, you will not be able to refer to the fields in the repeated segments at the bottom of the join structure.
- If you use tag names in a recursive joined structure, note the following guidelines:
  - If tag names are specified in a recursive join, duplicate field names must be qualified with the tag name.
  - If a join name is specified and tag names are not specified in a recursive join, duplicate field names must be prefixed with the first four characters of the join name.
  - If both a join name and a tag name are specified in a recursive join, the tag name must be used as a qualifier.
  - The tag name must be used as the field name qualifier in order to retrieve duplicate field names in a non-recursive join. If you do not qualify the field name, the first occurrence is retrieved.
- You may use a DEFINE-based join (see *How to Join From a Virtual Field to a Real Field* on page 13-17) to join a virtual field in a descendant segment to a field in the parent segment.
- You can extend a recursive structure by issuing multiple JOIN commands from the bottom repeat segment in the structure to the parent segment, creating a structure up to 16 levels deep.
- For FOCUS data sources, the field in the parent segment to which you are joining must be indexed.
- For IMS data sources, the following applies:
  - The parent segment must be the root segment of the data source.
  - The field to which you are joining must be both a key field and a primary or secondary index.
  - You need a duplicate PCB in the PSB for every recursive join you create.

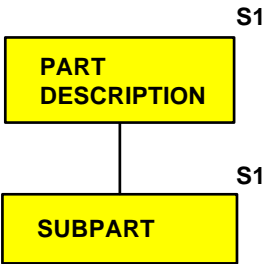
**Example**      **Using Recursive Joined Structures**

This example explains how to use recursive joins to store and report on a bill of materials. Suppose you are designing a data source called AIRCRAFT that contains the bill of materials for a company’s aircraft. The data source records data on three levels of airplane parts:

- Major divisions, such as the cockpit or cabin.
- Parts of divisions, such as instrument panels and seats.
- Subparts, such as nuts and bolts.

The data source must record each part, the part description, and the smaller parts composing the part. Some parts, such as nuts and bolts, are common throughout the aircraft. If you design a three-segment structure, one segment for each level of parts, descriptions of common parts will be repeated every place they are used.

To reduce this repetition, design a data source that has only two segments (shown in the following diagram). The top segment describes each part of the aircraft, large and small. The bottom segment lists the component parts without descriptions.

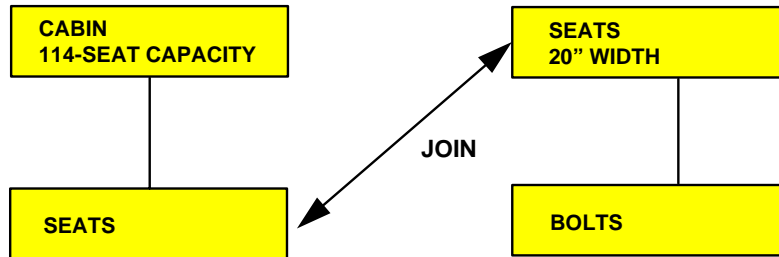


**AIRCRAFT Data Source**

Every part (except the largest divisions) appears in both the top segment, where it is described, and in the bottom segment, where it is listed as one of the components of a larger part. (The smallest parts, such as nuts and bolts, appear in the top segment without an instance of a child in the bottom segment.) Note that each part, no matter how often it is used in the aircraft, is described only once.

If you join the bottom segment to the top segment, the descriptions of component parts in the bottom segment can be retrieved. The first-level major divisions can also be related to third-level small parts, going from the first-level to the second-level to the third-level. Thus, the structure behaves as a three-level data source, although it is actually a more efficient two-level.

For example, CABIN is a first-level division appearing in the top segment. It lists SEATS as a component in the bottom segment. SEATS also appears in the top segment; it lists BOLTS as a component in the bottom segment. If you join the bottom segment to the top segment, you can go from CABIN to SEATS and from SEATS to BOLTS.

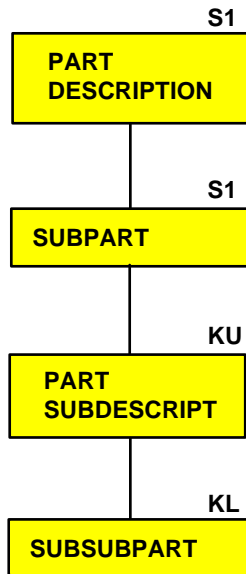


**Sample Instance in the AIRCRAFT Data Source**

Join the bottom segment to the top segment with this JOIN command:

```
JOIN SUBPART IN AIRCRAFT TO PART IN AIRCRAFT AS SUB
```

This creates the following recursive structure.



**AIRCRAFT Data Source  
Recursively Joined**

You can then produce a report on all three levels of data with this TABLE command (the field SUBDESCRIPT describes the contents of the field SUBPART):

```
TABLE FILE AIRCRAFT
PRINT SUBSUBPART BY PART BY SUBPART BY SUBDESCRIPT
END
```

## How the JOIN Command Works

The JOIN command enables you to report from two or more related data sources with a single request. Joined data sources remain physically separate, but are treated as one. Up to 16 joins can be in effect at any one time. The join applies to all requests run during the session in which it is issued, unless it is explicitly cleared.

When two data sources are joined, one is called the *host* file; the other is called the *cross-referenced* file. Each time a record from the host file is retrieved, the corresponding fields in the cross-referenced file are identified if they are referenced in the report request. The records in the cross-referenced file containing the corresponding values are then retrieved.

Two data sources can be joined using a conditional join whenever you can define an expression that determines how to relate records in the host file to records in the cross-referenced file. Two data sources can be joined using an equijoin when they have two fields—one in each data source—with formats (character, numeric, or date) and values in common. The common formats ensure the proper interpretation of the values. For example, suppose that you need to read data from two data sources: one named JOB, containing job information, and a second named SALARY, containing salary information. You can join these two data sources if each has a field identifying the same group of employees in the same way: by last name, serial number, or social security number. The join becomes effective when common values (let's say common social security numbers) are retrieved for the joined fields.

After you issue the JOIN command, you can issue a single TABLE, TABLEF, MATCH FILE, or GRAPH request to read the joined data source. You only need to specify the first data source (host file) to produce a report from two or more data sources. For example, assume you are writing a report from the JOB and SALARY data sources. You execute the following equijoin:

```
JOIN EMP_ID IN JOB TO ALL EMP_ID IN SALARY
```

This command joins the field EMP\_ID in the JOB file to the field EMP\_ID in the SALARY file. JOB is the host file and SALARY is the cross-referenced file. You then execute this report request:

```
TABLE FILE JOB
PRINT SALARY AND JOB_TITLE BY EMP_ID
END
```

The first record retrieved is a JOB file record for employee #071382660. Next, all records in the SALARY data source containing employee #071382660 are retrieved. This process continues until all the records have been read.

You can base your join on:

- Real fields that have been declared in the Master Files of the host and cross-referenced data sources, respectively. See *How to Join Real Fields* on page 13-13.
- A virtual field in the host data source (that has either been defined in the Master File or with a DEFINE command) and a real field that has been declared in the Master File of the cross-referenced data source. See *How to Join From a Virtual Field to a Real Field* on page 13-17.

**Reference****Increasing Retrieval Speed in Joined Data Sources**

You can increase retrieval speed in joined structures by using an external index. However, the target segment for the index cannot be a cross-referenced segment. For related information, see Chapter 15, *Improving Report Processing*.

## Creating an Equijoin

The most common joined structures are based on real fields that have been declared in the Master Files of the host and cross-referenced data sources, respectively.

**Syntax****How to Join Real Fields**

The following JOIN syntax requires that the fields you are using to join the files are real fields declared in the Master File. This may be a simple join based on one field in each file to be joined or a multi-field join for data sources that support this behavior. The following syntax describes the simple and multi-field variations

```
JOIN field1 [AND field1a...] IN host [TAG tag1]
TO [ALL] field2 [AND field2a...] IN crfile [TAG tag2] [AS joinname]
END
```

where:

`JOIN field1`

Is the name of a field in the host file that contains values shared with a field in the cross-referenced file. This field is called the host field.

`AND field1a...`

Can be an additional field in the host file with the caveats noted below. The phrase beginning with AND is required when specifying multiple fields.

- When you are joining two FOCUS data sources you can specify up to four alphanumeric fields in the host file that, if concatenated, contain values shared with the cross-referenced file. You may not specify more than one field in the cross-referenced file when the suffix of the cross-referenced file is FOC. For example, assume the cross-referenced file contains a phone number field having an area code-prefix-exchange format. The host file has an area code field, a prefix field, and an exchange field. You can specify these three fields to join them to the phone number field in the cross-referenced file. The JOIN command treats the three fields as one. Other data sources do not have this restriction on the cross-referenced file.
- For data adapters that support multi-field and concatenated joins, you can specify up to 16 fields. See your data adapter documentation for specific information about supported join features. Note that FOCUS data sources do not support these joins.

`IN host`

Is the name of the host file.

**TAG** *tag1*

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in host files.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

**TO** [ALL] *field2*

Is the name of a field in the cross-referenced file that contains values that match those of *field1* (or of concatenated host fields). This field is called the cross-referenced field. For related information see *Requirements for Cross-Referenced Fields in an Equijoin* on page 13-15.

Use the ALL parameter when *field2* may have multiple instances in common with one value in *field1*. This is called a non-unique (one-to-many) join. See *Unique and Non-Unique Joined Structures* on page 13-4.

**AND** *field2a...*

*Field2a* is the name of a field in the cross-referenced file with values in common with *field1a*.

**Note:** Field2a may be qualified. This field is only available for data adapters that support multi-field joins.

**IN** *crfile*

Is the name of the cross-referenced file.

**TAG** *tag2*

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in cross-referenced files. In a recursive join structure, if no tag name is provided, all field names and aliases are prefixed with the first four characters of the join name. For related information, see *Usage Notes for Recursive Joined Structures* on page 13-9.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

**AS** *joinname*

Is an optional name of up to eight characters that you may assign to the join structure. You must assign a unique name to a join structure if:

- You want to ensure that a subsequent JOIN command will not overwrite it.
- You want to clear it selectively later.
- The structure is recursive, as explained in *Recursive Joined Structures* on page 13-7.

**Note:** If you do not assign a name to the join structure with the AS phrase, the name is assumed to be blank. A join without a name will overwrite an existing join without a name.

**END**

Required when the JOIN command is longer than one line; terminates the command.



## Example

### Creating a Simple Unique Joined Structure

An example of a simple unique join is shown below:

```
JOIN JOBCODE IN EMPLOYEE TO JOBCODE IN JOBFILE AS JJJOIN
```

## Reference

### Requirements for Cross-Referenced Fields in an Equijoin

The cross-referenced fields used in a JOIN must have the following characteristics in specific data sources:

- In relational data sources and in a CA-DATACOM/DB data source, the cross-referenced field can be any field.
- In FOCUS and Fusion data sources, the cross-referenced field must be indexed. Indexed fields have the attribute FIELDTYPE=I or INDEX=I or INDEX=ON in the Master File. If the cross-referenced field does not have this attribute, append the attribute to the field declaration in the Master File and rebuild the file using the REBUILD utility with the INDEX option. This will add an index to your FOCUS or Fusion data source.

**Note:** The indexed fields can be external. For more information, see the *Maintaining Databases* manual on REBUILD.

- In IMS data sources, the cross-referenced field must be a key field in the root segment. It can be a primary or secondary index.
- In fixed sequential files, the cross-referenced field can be any field. However, both the host and cross-referenced file must be retrieved in ascending order on the named (key) field. If the data is not in the same sort order, errors will be displayed. If the cross-referenced file contains only one segment, the host file must have a segment declaration.

## Reference

### Restrictions on Group Fields

When group fields are used in a joined structure, the group in the host file and the group in the cross-referenced file must have the same number of elements.

- In ISAM data sources, the field must be the full primary key if you issue a unique join or an initial subset of the primary key if you issue a non-unique join. In the Master File, the primary key is described by a key GROUP; the initial subset is the first field in that group.
- In VSAM KSDS data sources, the field must be the full primary or alternate key if you issue a unique join or an initial subset of the primary or alternate key if you issue a non-unique join. In the Master File, the primary key is described by a key GROUP; the initial subset is the first field in that group.

In VSAM ESDS data sources, the field can be any field, as long as the file is already sorted on that field.

- In Model 204 data sources, the field must be a key field. In the Access File, the types of key fields are alphanumeric (KEY), ordered character (ORA), ordered numeric (ORN), numeric range (RNG), invisible (IVK), and invisible range (IVR).
- In ADABAS data sources, the field must be a descriptor field, a superdescriptor defined with the .SPR or .NOP field name suffix, or a subdescriptor defined with the .NOP field name suffix. The field description in the Master File must contain the attribute FIELDTYPE=I.

In the Access File, the cross-referenced segment must specify ACCESS=ADBS and either CALLTYPE=Find or CALLTYPE=RL. If CALLTYPE=RL, the host field can be JOINed to the high-order portion of a descriptor, superdescriptor, or subdescriptor, if the high-order portion is longer than the host field.

- In CA-IDMS/DB data sources, the field must be an indexed field on a network record, identified by the attribute FIELDTYPE=I in the Master File; a CA-IDMS/DB CALC field on a network record, identified by the CLCFLD phrase in the Access File; or any field on an LRF or ASF record.

## Joining From a Virtual Field to a Real Field Using an Equijoin

You can use DEFINE-based JOIN syntax to create a virtual host field that you can join to a real cross-referenced field. The DEFINE expression that creates the virtual host field may contain only fields in the host file and constants. (It may not contain fields in the cross-referenced file.) You can do more than one join from a virtual field.

You can create the virtual host field in a separate DEFINE command or in a Master File. For information on Master Files see the *Describing Data* manual.

The same report request can use JOIN-based virtual fields and virtual fields unrelated to the join.

Note that if you are creating a virtual field in a DEFINE command, you must issue the DEFINE after the JOIN command, but before the TABLE request since a JOIN command clears all fields created by DEFINE commands for the host file and the joined structure. Virtual fields defined in Master Files are not cleared.

### Tip:

If a DEFINE command precedes the JOIN command, you can set KEEPDEFINES ON to reinstate virtual fields during the parsing of a subsequent JOIN command. For more information see *Preserving Virtual Fields Using KEEPDEFINES* on page 13-26.

## Syntax

### How to Join From a Virtual Field to a Real Field

The DEFINE-based JOIN command enables you to join a virtual field in the host file to a real field in the cross-referenced file. The syntax is

```
JOIN deffld WITH host_field IN hostfile [TAG tag1]
TO [ALL] cr_field IN crfile [TAG tag2] [AS joinname]
END
```

where:

`JOIN deffld`

Is the name of the virtual field for the host file (the host field). The virtual field can be defined in the Master File or with a DEFINE command. For related information, see *Notes on Using Virtual Fields With Joined Data Sources* on page 13-19.

`WITH host_field`

Is the name of any real field in the host segment with which you want to associate the virtual field. This association is required to locate the virtual field.

To determine which segment contains the virtual field, use the ? DEFINE query. See the *Developing Applications* manual for details about Query commands.

`IN hostfile`

Is the name of the host file.

**TAG tag1**

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in host files.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

**TO cr\_field**

Is the name of a real field in the cross-referenced data source whose values match those of the virtual field. This must be a real field declared in the Master File.

**ALL**

Is the parameter to use when the cross-referenced field may have multiple values in common with one value in the virtual host field. See *Unique and Non-Unique Joined Structures* on page 13-4 for more information.

**IN crfile**

Is the name of the cross-referenced file.

**TAG tag2**

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in cross-referenced files. In a recursive joined structure, if no tag name is provided, all field names and aliases are prefixed with the first four characters of the join name. For related information, see *Usage Notes for Recursive Joined Structures* on page 13-9.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

**AS joinname**

Is an optional name of up to eight characters that you may assign to the joined structure. You must assign a unique name to a join structure if:

- You want to ensure that a subsequent JOIN command will not overwrite it.
- You want to clear it selectively later.
- The structure is recursive, and you do not specify tag names. See *Recursive Joined Structures* on page 13-7.

If you do not assign a name to the joined structure with the AS phrase, the name is assumed to be blank. A join without a name will overwrite an existing join without a name.

**END**

Required when the JOIN command is longer than one line; terminates the command.

## Reference

### Notes on Using Virtual Fields With Joined Data Sources

Requests reading joined data sources can contain virtual fields that are defined either:

- In the Master File of the host data source.
- In a DEFINE command, in which the syntax

```
DEFINE FILE hostfile
```

identifies the host data source in the joined structure.

**Note:** The expression defining the host field for the join can use only host fields and constants.

All other virtual fields can contain real fields from the host file and the cross-referenced file.

#### Tip:

Since issuing the JOIN command clears all DEFINE commands for the host file and the joined structure, you must issue the DEFINE command after the JOIN or you can turn KEEPDEFINES ON to preserve the virtual fields. For more information see *Preserving Virtual Fields During Join Parsing* on page 13-26.

**Example**                      **Creating a Virtual Host Field for a Joined Structure**

Suppose that a retail chain sends four store managers to attend classes. Each person, identified by an ID number, manages a store in a different city. The stores and the cities where they are located are contained in the SALES data source. The manager IDs, the classes, and dates the managers attended are contained in the EDUCFILE data source.

The following procedure lists the courses that the managers attended, identifying the managers by the cities where they work. Note the three elements in the procedure:

- The JOIN command joins the SALES data source to the EDUCFILE data source, based on the values common to the ID\_NUM field (which contains manager IDs) in SALES and the EMP\_ID field in EDUCFILE. Note that the ID\_NUM field does not exist yet; it will be created by the DEFINE command.
- The DEFINE command creates the ID\_NUM field, assigning to it the IDs of the managers working in the four cities.
- The TABLE command produces the report.

The procedure is:

```
JOIN ID_NUM WITH CITY IN SALES TO ALL EMP_ID IN EDUCFILE AS SALEDUC

DEFINE FILE SALES
ID_NUM/A9 = DECODE CITY ('NEW YORK' 451123478 'NEWARK' 119265415
 'STAMFORD' 818692173 'UNIONDALE' 112847612);

END

TABLE FILE SALES
PRINT DATE_ATTEND BY CITY BY COURSE_NAME
END
```

The output is:

| CITY      | COURSE_NAME                | DATE_ATTEND |
|-----------|----------------------------|-------------|
| ----      | -----                      | -----       |
| NEW YORK  | FILE DESCRIPT & MAINT      | 81/11/15    |
| NEWARK    | BASIC RPT NON-DP MGRS      | 82/08/24    |
| STAMFORD  | BASIC REPORT PREP DP MGRS  | 82/08/02    |
|           | HOST LANGUAGE INTERFACE    | 82/10/21    |
| UNIONDALE | BASIC REPORT PREP FOR PROG | 81/11/16    |
|           | FILE DESCRIPT & MAINT      | 81/11/15    |

## Data Formats of Shared Fields

Generally, the fields containing the shared values in the host and cross-referenced files must have the same data formats.

If you specify multiple host file fields, the JOIN command treats the fields as one concatenated field. Add the field format lengths to obtain the length of the concatenated field. You must observe the following rules:

- If the host field is an alphanumeric field, the cross-referenced field must also be alphanumeric and have the same length.

The formats may have different edit options.

Note that a text field cannot be used to join data sources.

- If the host field is a numeric field, the host field format, as specified by the USAGE (or FORMAT) attribute in the Master File, must agree in type (I, P, F, or D) with the format of the cross-referenced field as specified by the USAGE (or FORMAT) attribute, unless you issue the SET JOINOPT command, which enables you to join numeric fields with different data types. For details, see *Joining Fields With Different Numeric Data Types* on page 13-22.

The edit options may differ. The length may also differ, but with the following effect:

- If the format of the host field (as specified by the USAGE attribute) is packed decimal (P) or integer (I) and is longer than the cross-referenced field format (specified by the USAGE attribute for FOCUS data sources or the ACTUAL attribute for other data sources, only the length of the cross-referenced field format is compared, using only the right-most digits of the shorter field. For example, if a five-digit packed decimal format field is joined to a three-digit packed decimal format field, when a host record with a five-digit number is retrieved, all cross-referenced records with the last three digits of that number are also retrieved.
- If the format of the host field is double precision (D), the left-most eight bytes of each field are compared.
- If the host field is a date field, the cross-referenced field must also be a date field. Date and date-time fields must have the same components, not necessarily in the same order.
- The host and cross-referenced fields can be described as groups in the Master File if they contain the same number of component fields. The corresponding component fields in each group (for example, the first field in the host group and the first field in the cross-referenced group) must obey the above rules. For related information, see *Restrictions on Group Fields* on page 13-16.

If the host field is not a group field, the cross-referenced field can be a group. If the host field is a group, the cross-referenced field must also be a group.

## Joining Fields With Different Numeric Data Types

You can join two or more data sources containing different numeric data types. For example, you can join a field with a short packed decimal format to a field with a long packed decimal format, or a field with an integer format to a field with a packed decimal format. This provides enormous flexibility for creating reports from joined data sources.

- When joining a shorter field to a longer field, the cross-referenced value is padded to the length of the host field, adding spaces (for alpha fields) or hexadecimal zeros (for numeric fields). This new value is used for searches in the cross-referenced file.
- When joining a longer field to a shorter field, the FROM value is truncated. If part of your value is truncated due to the length of the USAGE in the cross-referenced file, only records matching the truncated value will be found in the cross-referenced file.

**Note:** For comparison on packed decimal fields to be accomplished properly, all signs for positive values are converted to hex C and all signs for negative values are converted to hex D.

### Syntax

### How to Enable Joins With Data Type Conversion

To enable joins with data type conversion, issue the command

```
SET JOINOPT = [NEW|OLD]
```

where:

**NEW**

Enables joins with data type conversion.

**OLD**

Disables joins with data type conversion. This value is the default.

### Example

### Issuing Joins With Data Type Conversion

Since you can join a field with a short packed decimal format to a field with a long packed decimal format, a join can be defined in the following Master Files:

```
FILE=PACKED,SUFFIX=FIX,$
 SEGNAME=ONE,SEGTYPE=S0
 FIELD=FIRST,,P8,P4,INDEX=I,$

FILE=PACKED2,SUFFIX=FIX,$
 SEGNAME=ONE,SEGTYPE=S0
 FIELD=PFIRST,,P31,P16,INDEX=I,$
```

The JOIN command might look like this:

```
JOIN FIRST IN PACKED TO ALL PFIRST IN PACKED2 AS J1
```

When joining packed fields, the preferred sign format of X'C for positive values and X'D' for negative values is still required. All other non-preferred signs are converted to either X'C' or X'D'.



# Using a Conditional Join

Using conditional JOIN syntax, you can establish joins based on conditions other than equality between fields. In addition, the host and cross-referenced join fields do not have to contain matching formats, and the cross-referenced field does not have to be indexed.

The conditional join is supported for FOCUS and for VSAM, ADABAS, IMS, and all relational data sources. Because each data source differs in its ability to handle complex WHERE criteria, the optimization of the conditional JOIN syntax differs depending on the specific data sources involved in the join and the complexity of the WHERE criteria.

The standard ? JOIN command lists every join currently in effect, and indicates any that are based on WHERE criteria. For details and sample output, see the *Developing Applications* manual.

## Syntax

### How to Create a Conditional JOIN

The syntax of the conditional (WHERE-based) JOIN command is

```
JOIN FILE from_file AT from_field [TAG from_tag] [WITH fieldname]
 TO [ALL|ONE]
 FILE to_file AT to_field [TAG to_tag]
 [AS as_name]
 [WHERE expression1 ;
 WHERE expression2 ;
 ... ;]
END
```

where:

*from\_file*

Is the host Master File.

AT

Links the correct parent segment or host to the correct child or cross-referenced segment. The field values used as the AT parameter are not used to cause the link. They are simply used as segment references.

*from\_field*

Is the field name in the host Master File whose segment will be joined to the cross-referenced data source. The field name must be at the lowest level segment in its data source that will be referenced.

*from\_tag*

Is the optional tag name that is used as a unique qualifier for fields and aliases in the host data source.

*fieldname*

Is a data source field with which to associate a DEFINE-based conditional join. For a DEFINE-based conditional join, the KEEPDEFINES setting must be on and you must create the virtual fields before issuing the JOIN command.

ALL

Is the one-to-many relationship between the *from\_file* and *to\_file*.

**ONE**

Is the one-to-one relationship between the *from\_file* and *to\_file*.

**to\_file**

Is the cross-referenced Master File.

**to\_field**

Is the join field name in the cross-referenced Master File. It can be any field in the segment.

**to\_tag**

Is the optional tag name that is used as a unique qualifier for fields and aliases in the cross-referenced data source.

**as\_name**

Is the name associated with the joined structure.

**expression1, expression2**

Are any expression that is acceptable in a DEFINE FILE command. All fields used in the expression must lie on a single path.

**Note:** Single line JOIN syntax is not supported. The END command is required.

## Example

### Using a Conditional Join

The following example joins the VIDEOTRK and MOVIES data sources on the conditions that:

- The transaction date (in VIDEOTRK) is more than ten years after the release date (in MOVIES).
- The movie codes match in both data sources.

The join is performed at the segment that contains MOVIECODE in the VIDEOTRK data source because the join must occur at the lowest segment referenced.

The following request displays the title, most recent transaction date, and release date for each movie in the join and computes the number of years between this transaction date and the release date:

```
JOIN FILE VIDEOTRK AT MOVIECODE TAG V1 TO ALL
 FILE MOVIES AT RELDATE TAG M1 AS JW1
 WHERE DATEDIF(RELDATE , TRANSDATE,'Y') GT 10 ;
 WHERE V1.MOVIECODE EQ M1.MOVIECODE;
END
TABLE FILE VIDEOTRK
 SUM TITLE/A25 AS 'Title'
 TRANSDATE AS 'Last,Transaction'
 RELDATE AS 'Release,Date'
 COMPUTE YEARS/I5 = (TRANSDATE - RELDATE)/365; AS 'Years,Difference'
 BY TITLE NOPRINT
 BY HIGHEST 1 TRANSDATE NOPRINT
END
```

The output is:

| Title                   | Last<br>Transaction | Release<br>Date | Years<br>Difference |
|-------------------------|---------------------|-----------------|---------------------|
| -----                   | -----               | -----           | -----               |
| ALICE IN WONDERLAND     | 91/06/22            | 51/07/21        | 39                  |
| ALIEN                   | 91/06/18            | 80/04/04        | 11                  |
| ALL THAT JAZZ           | 91/06/25            | 80/05/11        | 11                  |
| ANNIE HALL              | 91/06/24            | 78/04/16        | 13                  |
| BAMBI                   | 91/06/22            | 42/07/03        | 49                  |
| BIRDS, THE              | 91/06/23            | 63/09/27        | 27                  |
| CABARET                 | 91/06/25            | 73/07/14        | 17                  |
| CASABLANCA              | 91/06/27            | 42/03/28        | 49                  |
| CITIZEN KANE            | 91/06/22            | 41/08/11        | 49                  |
| CYRANO DE BERGERAC      | 91/06/20            | 50/11/09        | 40                  |
| DEATH IN VENICE         | 91/06/26            | 73/07/27        | 17                  |
| DOG DAY AFTERNOON       | 91/06/23            | 76/04/04        | 15                  |
| EAST OF EDEN            | 91/06/20            | 55/01/12        | 36                  |
| GONE WITH THE WIND      | 91/06/24            | 39/06/04        | 52                  |
| JAWS                    | 91/06/27            | 78/05/13        | 13                  |
| MALTESE FALCON, THE     | 91/06/19            | 41/11/14        | 49                  |
| MARTY                   | 91/06/19            | 55/10/26        | 35                  |
| NORTH BY NORTHWEST      | 91/06/21            | 59/02/09        | 32                  |
| ON THE WATERFRONT       | 91/06/24            | 54/07/06        | 36                  |
| PHILADELPHIA STORY, THE | 91/06/21            | 40/05/06        | 51                  |
| PSYCHO                  | 91/06/17            | 60/05/16        | 31                  |
| REAR WINDOW             | 91/06/17            | 54/12/15        | 36                  |
| SHAGGY DOG, THE         | 91/06/25            | 59/01/09        | 32                  |
| SLEEPING BEAUTY         | 91/06/24            | 75/08/30        | 15                  |
| TIN DRUM, THE           | 91/06/17            | 80/03/01        | 11                  |
| VERTIGO                 | 91/06/27            | 58/11/25        | 32                  |

## Preserving Virtual Fields During Join Parsing

There are two ways to preserve virtual fields during join parsing. One way is to use `KEEPDEFINES`, and the second is to use `DEFINE FILE SAVE` and `DEFINE FILE RETURN`.

### Preserving Virtual Fields Using `KEEPDEFINES`

The `KEEPDEFINES` parameter determines if a virtual field created by the `DEFINE` command for a host or joined structure is retained or cleared after the `JOIN` command is run. It applies when the `DEFINE` command precedes the `JOIN` command.

The prior virtual fields constitute what is called a context. Each new context creates a new layer or command environment. When you first enter the new environment, all virtual fields defined in the previous layer are available in the new layer. Overwriting or clearing a virtual field definition affects only the current layer. When you return to the previous layer, its virtual field definitions are intact.

New `DEFINE` fields issued after the `JOIN` command constitute another context, and by so doing generate a stack of contexts. In each context, all virtual fields of all prior contexts are accessible.

- By default the `KEEPDEFINES` setting is `OFF`. With this setting, a `JOIN` command removes prior virtual fields.
- When `KEEPDEFINES` is set to `ON`, virtual fields are reinstated during the parsing of a subsequent `JOIN` command.

A `JOIN CLEAR as_name` command will remove all the contexts that were created after the `JOIN as_name` was issued.

For `DEFINE`-based conditional joins, the `KEEPDEFINES` setting must be `ON`. You then must create all virtual fields before issuing the `DEFINE`-based conditional `JOIN` command. This differs from traditional `DEFINE`-based joins in which the virtual field is created after the `JOIN` command. In addition, a virtual field may be part of the `JOIN` syntax or `WHERE` syntax.

`DEFINE` commands issued after the `JOIN` command do not replace or clear the virtual fields created before the join since a new file context is created.

### Syntax

#### How to Use `KEEPDEFINES`

```
SET KEEPDEFINES = {ON|OFF}
```

where:

`ON`

Retains the virtual field after a `JOIN` command is run.

`OFF`

Clears the virtual field after a `JOIN` command is run. This value is the default.

## Example

### Preserving Virtual Fields During Join Parsing With KEEPDEFINES

The first virtual field, DAYSKEPT, is defined prior to issuing any joins, but after setting KEEPDEFINES to ON. DAYSKEPT is the number of days between the return date and rental date for videotape:

```
SET KEEPDEFINES = ON
DEFINE FILE VIDEOTRK
DAYSKEPT/I5 = RETURNDATE - TRANSDATE;
END
```

The ? DEFINE query command shows that this is the only virtual field defined at this point:

```
? DEFINE

FILE FIELD NAME FORMAT SEGMENT VIEW TYPE
VIDEOTRK DAYSKEPT I5 4
```

The following request prints all transactions in which the number of days kept is two:

```
TABLE FILE VIDEOTRK
PRINT MOVIECODE TRANSDATE RETURNDATE DAYSKEPT
WHERE DAYSKEPT EQ 2
END
```

The first few lines of output show that each return date is two days after the transaction date:

| MOVIECODE | TRANSDATE | RETURNDATE | DAYSKEPT | ACTUAL_DAYS |
|-----------|-----------|------------|----------|-------------|
| -----     | -----     | -----      | -----    | -----       |
| 001MCA    | 91/06/27  | 91/06/29   | 2        | 2           |
| 692PAR    | 91/06/27  | 91/06/29   | 2        | 2           |
| 259MGM    | 91/06/19  | 91/06/21   | 2        | 2           |

Now, the VIDEOTRK data source is joined to the MOVIES data source. The ? DEFINE query shows that the join did not clear the DAYSKEPT virtual field:

```
JOIN MOVIECODE IN VIDEOTRK TO ALL MOVIECODE IN MOVIES AS J1
? DEFINE

FILE FIELD NAME FORMAT SEGMENT VIEW TYPE
VIDEOTRK DAYSKEPT I5 4
```

Next a new virtual field, YEARS is defined for the join between VIDEOTRK and MOVIES:

```
DEFINE FILE VIDEOTRK
YEARS/I5 = (TRANSDATE - RELDATE)/365;
END
```

The ? DEFINE query shows that the virtual field created prior to the join was not cleared by this new virtual field because it was in a separate context:

```
? DEFINE

FILE FIELD NAME FORMAT SEGMENT VIEW
TYPE
VIDEOTRK DAYSKEPT I5 4
VIDEOTRK YEARS I5 5
```

Next, the field DAYSKEPT is re-defined so that it is the actual number of days plus one:

```
DEFINE FILE VIDEOTRK
DAYSKEPT/I5 = RETURNDATE - TRANSDATE + 1;
END
```

The ? DEFINE query shows that there are two versions of the DAYSKEPT virtual field. However, YEARS was cleared because it was in the same context (after the join) as the new version of DAYSKEPT and the DEFINE command did not specify the ADD option:

```
? DEFINE

FILE FIELD NAME FORMAT SEGMENT VIEW
TYPE
VIDEOTRK DAYSKEPT I5 4
VIDEOTRK DAYSKEPT I5 4
```

The same request now uses the new definition for DAYSKEPT. Note that the number of days between the return date and transaction date is actually one day, not two because of the change in the definition of DAYSKEPT:

| MOVIECODE | TRANSDATE | RETURNDATE | DAYSKEPT | ACTUAL_DAYS |
|-----------|-----------|------------|----------|-------------|
| -----     | -----     | -----      | -----    | -----       |
| 040ORI    | 91/06/20  | 91/06/21   | 2        | 1           |
| 505MGM    | 91/06/21  | 91/06/22   | 2        | 1           |
| 710VES    | 91/06/26  | 91/06/27   | 2        | 1           |

Now, J1 is cleared. The redefinition for DAYSKEPT is also cleared:

```
JOIN CLEAR J1
? DEFINE

FILE FIELD NAME FORMAT SEGMENT VIEW
TYPE
VIDEOTRK DAYSKEPT I5 4
```

The report output shows that the original definition for DAYSKEPT is now in effect:

| MOVIECODE | TRANSDATE | RETURNDATE | DAYSKEPT | ACTUAL_DAYS |
|-----------|-----------|------------|----------|-------------|
| -----     | -----     | -----      | -----    | -----       |
| 001MCA    | 91/06/27  | 91/06/29   | 2        | 2           |
| 692PAR    | 91/06/27  | 91/06/29   | 2        | 2           |
| 259MGM    | 91/06/19  | 91/06/21   | 2        | 2           |

# Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN

The DEFINE FILE SAVE command forms a new context for virtual fields, which can then be removed with DEFINE FILE RETURN.

For details, see Chapter 6, *Creating Temporary Fields*.

## Example

### Using DEFINE FILE SAVE and RETURN

The following command enables you to preserve virtual fields within a file context:

```
SET KEEPDEFINES=ON
```

The following command defines virtual field A for the VIDEOTRK data source and places it in the current context:

```
DEFINE FILE VIDEOTRK
 A/A5='JAWS' ;
END
```

The following command creates a new context and saves virtual field B in this context:

```
DEFINE FILE VIDEOTRK SAVE
 B/A5='ROCKY' ;
END
? DEFINE
```

The output of the ? DEFINE query lists virtual fields A and B:

| FILE       | FIELD NAME | FORMAT | SEGMENT | VIEW |
|------------|------------|--------|---------|------|
| TYPE       |            |        |         |      |
| VIDEOTRK A |            | A5     |         |      |
| VIDEOTRK B |            | A5     |         |      |

The following DEFINE command creates virtual field C. All previously defined virtual fields are cleared because the ADD option was not used in the DEFINE command:

```
DEFINE FILE VIDEOTRK
 C/A10='AIRPLANE' ;
END
? DEFINE
```

The output of the ? DEFINE query shows that C is the only virtual field defined:

| FILE       | FIELD NAME | FORMAT | SEGMENT | VIEW |
|------------|------------|--------|---------|------|
| TYPE       |            |        |         |      |
| VIDEOTRK C |            | A10    |         |      |

The following JOIN command creates a new context. Because KEEPDEFINES is set to ON, virtual field C is not cleared by the JOIN command:

```
JOIN MOVIECODE IN VIDEOTRK TAG V1 TO MOVIECODE IN MOVIES TAG M1 AS J1
? DEFINE
```

The output of the ? DEFINE query shows that field C is still defined:

| FILE     | FIELD NAME | FORMAT | SEGMENT | VIEW |
|----------|------------|--------|---------|------|
| TYPE     |            |        |         |      |
| VIDEOTRK | C          | A10    |         |      |

The next DEFINE command creates virtual field D in the new context created by the JOIN command:

```
DEFINE FILE VIDEOTRK SAVE
 D/A10='TOY STORY';
END
? DEFINE
```

The output of the ? DEFINE query shows that virtual fields C and D are defined:

| FILE     | FIELD NAME | FORMAT | SEGMENT | VIEW |
|----------|------------|--------|---------|------|
| TYPE     |            |        |         |      |
| VIDEOTRK | C          | A10    |         |      |
| VIDEOTRK | D          | A10    |         |      |

The DEFINE FILE RETURN command clears virtual field D created in the current context (after the JOIN):

```
DEFINE FILE VIDEOTRK RETURN
? DEFINE
```

The output of the ? DEFINE query shows that virtual field D was cleared, but C is still defined:

| FILE     | FIELD NAME | FORMAT | SEGMENT | VIEW |
|----------|------------|--------|---------|------|
| TYPE     |            |        |         |      |
| VIDEOTRK | C          | A10    |         |      |

The following DEFINE FILE RETURN command does not clear virtual field C because field C was not created using a DEFINE FILE SAVE command:

```
DEFINE FILE VIDEOTRK RETURN
END
? DEFINE
```

The output of the ? DEFINE query shows that virtual field C is still defined:

| FILE     | FIELD NAME | FORMAT | SEGMENT | VIEW |
|----------|------------|--------|---------|------|
| TYPE     |            |        |         |      |
| VIDEOTRK | C          | A10    |         |      |

**Note:** DEFINE FILE RETURN is only activated when a DEFINE FILE SAVE is in effect.



## Screening Segments With Conditional JOIN Expressions

The conditional JOIN command can reference any and all fields in the joined segment and any and all fields in the parent segment, or higher on the parent's path.

When active, these Join expressions screen the segment they reside on (the child or joined segment). That is, if no child segment passes the test defined by the expression, the join follows the rules of SET ALL=OFF, or SET ALL=ON when no child segment exists. Unlike WHERE phrases in TABLE commands, JOIN WHERE screening does not automatically screen the parent segment when SET ALL=ON.

## Parsing WHERE Criteria in a Join

WHERE criteria take effect in a join only when a TABLE request reference is made to a cross-referenced segment or its children. If no such reference is made, the WHERE has no effect.

The AT attribute is used to link the correct parent segment or host to the correct child or cross-referenced segment. The field values used as the AT parameter are not used to cause the link. They are simply used as segment references.

**Note:** If no WHERE criteria are in effect, you will receive a Cartesian product.

## Displaying Joined Structures

When you join two data sources together, they are subsequently treated as one logical structure. This structure results from appending the structure of the cross-referenced file to the structure of the host file. The segment in the cross-referenced file containing the shared value field becomes the child of the segment in the host file with the shared value field.

### Syntax

#### How to Display a Joined Structure

To display the joined structure, issue the following command

```
CHECK FILE hostfile PICTURE
```

where:

*hostfile*

Is the name of the host file. For an illustration, see *Displaying Joined Structures* on page 13-31.

## Example Displaying a Joined Structure

Notice that the segments belonging to the host file appear as regular segments outlined by asterisks; the segments belonging to the cross-referenced file appear as virtual segments outlined by dots. The segments of the cross-referenced file are also labeled with the cross-referenced file name below each segment.

```

JOIN PIN IN EMPDATA TO PIN IN SALHIST
CHECK FILE EMPDATA PICTURE
0 NUMBER OF ERRORS= 0
 NUMBER OF SEGMENTS= 2 (REAL= 1 VIRTUAL= 1)
 NUMBER OF FIELDS= 14 INDEXES= 1 FILES= 2
 NUMBER OF DEFINES= 1
 TOTAL LENGTH OF ALL FIELDS= 132
1SECTION 01.01
 STRUCTURE OF FOCUS FILE EMPDATA ON 03/05/01 AT 12.22.49
 EMPDATA
01 S1

*PIN **I
*LASTNAME **
*FIRSTNAME **
*MIDINITIAL **
* **

 I
 I
 I
 I SLHISTRY
02 I KU
.....
:PIN :K
:EFFECTDATE :
:OLDSALARY :
: :
: :
:.....:
 JOINED SALHIST

```

The top segment of the cross-referenced file structure is the one containing the shared-value field. If this segment is not the root segment, the cross-referenced file structure is inverted, as it would be in an alternate file view.

The cross-referenced file segment types in the joined structure are the following:

- In unique join structures, the top cross-referenced file segment has the segment type KU. Its unique child segments have segment type KLU; non-unique child segments have segment type KL.
- In non-unique join structures, the top cross-referenced file segment has the segment type KM. Its unique child segments have segment type KLU; non-unique child segments have segment type KL.

The host file structure remains unchanged. The cross-referenced file may still be used independently.

Syntax

How to List Joined Structures

To display a list of joined data sources, issue the following command:

? JOIN

This displays every JOIN command currently in effect. For example:

JOINS CURRENTLY ACTIVE

| HOST    |          |     | CROSSREFERENCE |         |     |    |     |    |
|---------|----------|-----|----------------|---------|-----|----|-----|----|
| FIELD   | FILE     | TAG | FIELD          | FILE    | TAG | AS | ALL | WH |
| -----   | ----     | --- | -----          | ----    | --- | -- | --- | -- |
| JOBCODE | EMPLOYEE |     | JOBCODE        | JOBFILE |     |    | N   | N  |

If the joined structure has no join name, the AS phrase is omitted. If two data sources are joined by multiple JOIN commands, only the first command you issued is displayed. The N in the WH column indicates that the join is not conditional. A Y indicates that the join is conditional.

## Clearing Joined Structures

You can clear specific join structures or all existing structures. Clearing deactivates the designated joins. If you clear a conditional join, all joins issued subsequently to that join that use the same host file also are cleared.

### Tip:

If you wish to list the current joins before clearing or see details about all active joined structures, issue the query command ? JOIN. For details and illustrations, see *How to List Joined Structures* on page 13-33.

### Syntax

#### How to Clear a Join

To clear a joined structure, issue this command

```
JOIN CLEAR {joinname|*}
```

where:

*joinname*

Is the AS name of the joined structure you want to clear.

\*

Clears all joined structures.

## Clearing a Conditional Join

You can clear a join by issuing the JOIN CLEAR command. The effect of the JOIN CLEAR command depends on whether any conditional join exists.

- If conditional joins are found, and were issued after the join you wish to clear, or if the join you wish to clear is a conditional join, then the JOIN CLEAR *as\_name* command removes all joins issued after the specified join.
- If no conditional joins were issued after the join you wish to clear, only the join you specify will be cleared. Any virtual fields saved in the context of a join that is cleared will also be cleared. Normal joins may or may not be cleared, depending on the position of the conditional join. The JOIN CLEAR \* command will clear every join issued, along with its associated virtual fields. However, all virtual fields in the null context will remain untouched.

**Note:** The null context is the context of the data source prior to any joins being issued.

## Example

## Clearing Joins

The following request creates three joins using VIDEOTRK as the host data source. The first two are conditional (JW1, JW2), and the third join is unconditional (J1):

```
JOIN FILE VIDEOTRK AT PRODCODE TO ALL
 FILE GGSales AT PCD AS JW1
WHERE PRODCODE NE PCD;
END
JOIN FILE VIDEOTRK AT TRANSDATE TO ALL
 FILE MOVIES AT RELDATE AS JW1
WHERE (TRANSDATE - RELDATE)/365 GT 10;
END
JOIN MOVIECODE IN VIDEOTRK TO MOVIECODE IN MOVIES AS J1
```

The next request creates a conditional join (JW3) using MOVIES as the host data source:

```
JOIN FILE MOVIES AT MOVIECODE TO ONE
 FILE VIDEOTRK AT TRANSDATE AS JW2
WHERE (TRANSDATE - RELDATE)/365 LT 2;
END
```

The last request creates a third conditional join (JW4) that uses VIDEOTRK as the host data source:

```
JOIN FILE VIDEOTRK AT LASTNAME TO ALL
 FILE EMPLOYEE AT LAST_NAME AS JW3
WHERE LASTNAME GE LAST_NAME;
END
```

Following is the output of the ? JOIN query after executing these joins:

```
? JOIN
JOINS CURRENTLY ACTIVE
```

| HOST      |          |     | CROSSREFERENCE |          |     |     |     |    |
|-----------|----------|-----|----------------|----------|-----|-----|-----|----|
| FIELD     | FILE     | TAG | FIELD          | FILE     | TAG | AS  | ALL | WH |
| ----      | ----     | --- | ----           | ----     | --- | --  | --- | -- |
| PRODCODE  | VIDEOTRK |     | PCD            | GGSales  |     | JW1 | Y   | Y  |
| TRANSDATE | VIDEOTRK |     | RELDATE        | MOVIES   |     | JW2 | Y   | Y  |
| MOVIECODE | VIDEOTRK |     | MOVIECODE      | MOVIES   |     | J1  | N   | N  |
| MOVIECODE | MOVIES   |     | TRANSDATE      | VIDEOTRK |     | JW3 | N   | Y  |
| LASTNAME  | VIDEOTRK |     | LAST_NAME      | EMPLOYEE |     | JW4 | Y   | Y  |

Clearing JW2 clears all joins that were issued after JW2 and that use the same host data source. JW1 remains because it was issued prior to JW2, and JW3 remains because it uses a different host data source:

```
JOIN CLEAR JW2
? JOIN
JOINS CURRENTLY ACTIVE
```

| HOST      |          |     | CROSSREFERENCE |          |     |     |     |    |
|-----------|----------|-----|----------------|----------|-----|-----|-----|----|
| FIELD     | FILE     | TAG | FIELD          | FILE     | TAG | AS  | ALL | WH |
| ----      | ----     | --- | ----           | ----     | --- | --  | --- | -- |
| PRODCODE  | VIDEOTRK |     | PCD            | GGSales  |     | JW1 | Y   | Y  |
| MOVIECODE | MOVIES   |     | TRANSDATE      | VIDEOTRK |     | JW3 | N   | Y  |

---

## CHAPTER 14

# Merging Data Sources

### Topics:

- Merging Data
- MATCH Processing
- MATCH Processing With Common High Order Sort Fields
- Fine Tuning MATCH Processing
- Universal Concatenation
- Merging Concatenated Data Sources
- Cartesian Product

You can gather data for your reports by merging the contents of data structures with the **MATCH** command, or concatenating data sources with the **MORE** phrase, and reporting from the combined data.

## Merging Data

You can merge two or more data sources, and specify which records will be merged and which will be screened out, using the MATCH command. The command creates a new data source—a HOLD file—into which it merges fields from the selected records. You can report from the new data source and use it as you would any other HOLD file. The merge process does not change the original data sources. For more information on HOLD files, see Chapter 11, *Saving and Reusing Report Output*.

You select the records to be merged into the new data source by specifying sort fields in the MATCH command. You specify one set of sort fields—using the BY phrase—for the first data source, and a second set of sort fields for the second data source. The MATCH command compares all sort fields that have been specified in common for both data sources, and then merges all records from the first data source whose sort values match those in the second data source into the new HOLD file. You can specify up to 32 sort sets; this includes the number of common sort fields.

In addition to merging data source records that share values, you can merge records based on other relationships. For example, you can merge all records whose values do not match—that is, all records in each data source whose sort values are not matched in the other data source. Yet another type of merge combines all records from the first data source with any matching records from the second data source.

You can merge up to six sets of data in one MATCH request—for example, you can merge six different data sources or data from the same data source up to six times.

## Syntax

### How to Merge Data Sources

The syntax of the MATCH command is similar to that of the TABLE command

```
MATCH FILE file1
.
.
.
RUN
FILE file2
.
.
.
[AFTER MATCH merge_phrase]
RUN
FILE file3
.
.
.
[AFTER MATCH merge_phrase]
END
```

where:

*file1*

Is the first data source from which MATCH retrieves requested records.

*merge\_phrase*

Specifies how the retrieved records from the files are to be compared. For details, see *Merge Phrases* on page 14-6.

*file2/file3*

Are additional data sources from which MATCH retrieves requested records.

Note that a RUN command must follow each AFTER MATCH command (except for the last one). The END command must follow the final AFTER MATCH command.

MATCH generates a single-segment HOLD file. You can print the contents of the HOLD file using the PRINT command with the wildcard character (\*). For related information, see *Merging Data Sources* on page 14-4.

## Reference

### Usage Notes for Merging Data Sources

- The ACROSS and WHERE TOTAL phrases, and the COMPUTE command, are not permitted in a MATCH request. You can, however, use the DEFINE command.
- A total of 32 BY phrases and the maximum number of display fields can be used in each MATCH request. The maximum number of display fields is determined by a combination of factors. For details, see Chapter 1, *Creating Tabular Reports*.
- Up to 32 sort sets are supported, including the number of common sort fields.
- You must specify at least one BY field for each file used in the MATCH request.
- When used with MATCH, the SET HOLDLIST command behaves as if HOLDLIST were set to ALL.
- You cannot use BY HIGHEST in a MATCH request.



Example                      Merging Data Sources

```
MATCH FILE EDUCFILE
SUM COURSE_CODE
BY EMP_ID
RUN
FILE EMPLOYEE
SUM LAST_NAME AND FIRST_NAME
BY EMP_ID BY CURR_SAL
AFTER MATCH HOLD OLD-OR-NEW
END
_*****
-* PRINT CONTENTS OF HOLD FILE
_*****
TABLE FILE HOLD
PRINT *
END
```

The merge phrase used in this example was OLD-OR-NEW. This means that records from both the first (old) data source plus the records from the second (new) data source will appear in the HOLD file.

Note that if your are working in an interactive environment, after you enter the command RUN, a message indicates how many records were retrieved and—if you are entering the MATCH request at the command line—prompts you for the name of the next data source to be merged.

The output is:

PAGE        1

| EMP_ID    | COURSE_CODE | CURR_SAL    | LAST_NAME | FIRST_NAME |
|-----------|-------------|-------------|-----------|------------|
| 071382660 | 101         | \$11,000.00 | STEVENS   | ALFRED     |
| 112847612 | 103         | \$13,200.00 | SMITH     | MARY       |
| 117593129 | 203         | \$18,480.00 | JONES     | DIANE      |
| 119265415 | 108         | \$9,500.00  | SMITH     | RICHARD    |
| 119329144 |             | \$29,700.00 | BANNING   | JOHN       |
| 123764317 |             | \$26,862.00 | IRVING    | JOAN       |
| 126724188 |             | \$21,120.00 | ROMANS    | ANTHONY    |
| 212289111 | 103         | \$ .00      |           |            |
| 219984371 |             | \$18,480.00 | MCCOY     | JOHN       |
| 315548712 | 108         | \$ .00      |           |            |
| 326179357 | 301         | \$21,780.00 | BLACKWOOD | ROSEMARIE  |
| 451123478 | 101         | \$16,100.00 | MCKNIGHT  | ROGER      |
| 543729165 |             | \$9,000.00  | GREENSPAN | MARY       |
| 818692173 | 302         | \$27,062.00 | CROSS     | BARBARA    |

# MATCH Processing

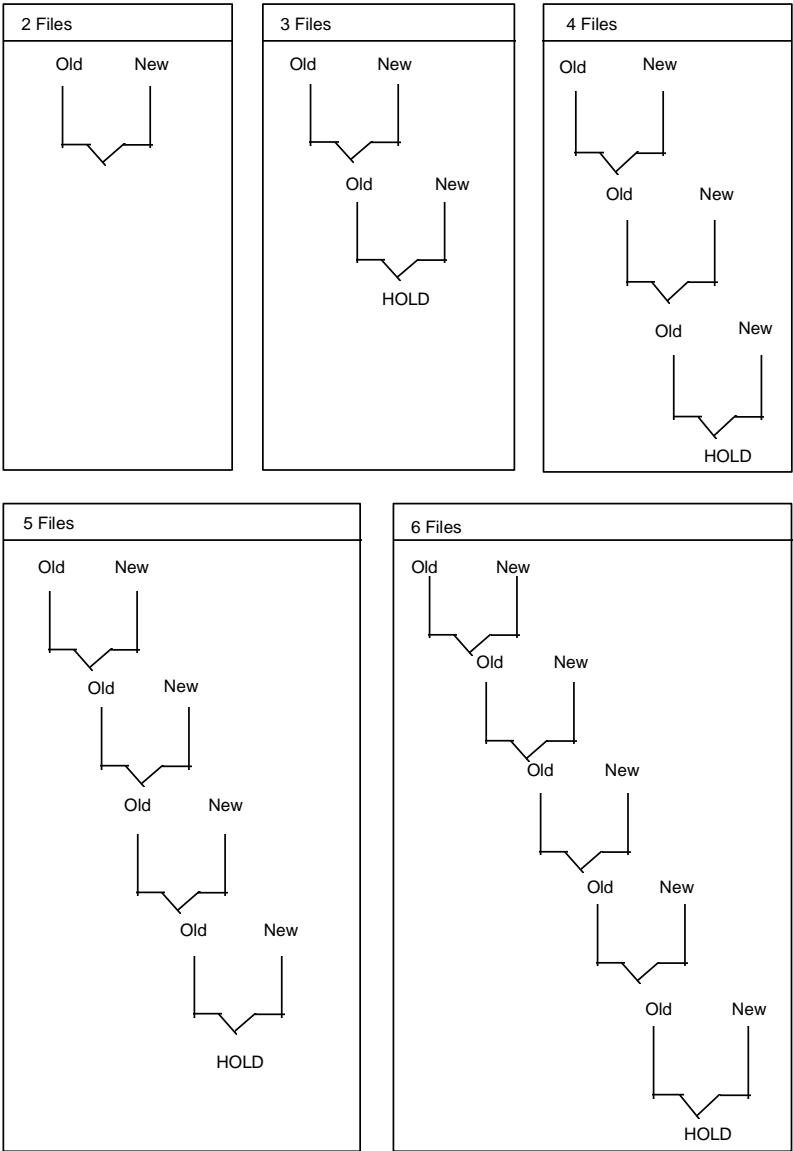
The way MATCH merges data depends on the order in which you name data sources in the request, the BY fields, display commands, and the merge phrases you use. In general, however, processing is as follows:

1. MATCH retrieves requested records from the first data source you name and writes them to a temporary work area.
2. MATCH retrieves requested records from the second data source you name and writes them to a temporary work area.
3. It compares the retrieved records' common high-order sort fields as specified in the merge phrase (for example, OLD-OR-NEW). For more information, see *Merge Phrases* on page 14-6.
4. It writes the merged results of the comparison to a temporary data source (if there are more MATCH operations). It cycles through all data sources named until END is encountered.
5. It writes final records to the HOLD file.

Reference Merge Phrases

MATCH logic depends on the concept of old and new data sources. Old refers to the first data source named in the request and new refers to the second data source. The result of each merge creates a HOLD file until the END command is encountered. The following diagram illustrates the general merge process:

The number of files to be merged



## Syntax

## How to Specify Merge Phrases

`AFTER MATCH HOLD [AS 'name'] mergetype`

where:

`AS 'name'`

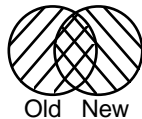
Specifies the name of the extract data source created by the MATCH command. The default is HOLD.

`mergetype`

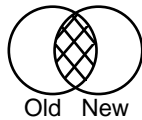
Specifies how the retrieved records from the files are to be compared.

The results of each phrase are graphically represented using Venn diagrams. In the diagrams, the left circle represents the old data source, the right circle represents the new data source, and the shaded areas represent the data that is written to the HOLD file.

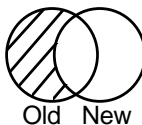
`OLD-OR-NEW` specifies that all records from both the old data source and the new data source will appear in the HOLD file. This is the default if the AFTER MATCH line is omitted.



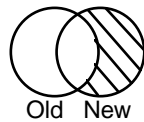
`OLD-AND-NEW` specifies that records that appear in both the old and new data sources appear in the HOLD file. (The intersection of the sets.)



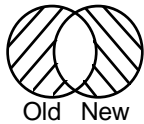
`OLD-NOT-NEW` specifies that records that appear only in the old data source will appear in the HOLD file.



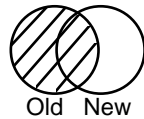
**NEW-NOT-OLD** specifies that records that appear only in the new data source will appear in the HOLD file.



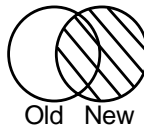
**OLD-NOR-NEW** specifies that only records that are in the old data source but not in the new data source, or in the new data source but not in the old, will appear in the HOLD file (the complete set of non-matching records from both data sources).



**OLD** specifies that all records from the old data source, and any matching records from the new data source, will be merged into the HOLD file.



**NEW** specifies that all records from the new data source, and any matching records from the old data source, will be merged into the HOLD file.



# MATCH Processing With Common High Order Sort Fields

When you construct your MATCH so that the first sort (BY) field (called the common high-order sort field) used for both data sources is the same, the match compares the values of the common high-order sort fields. If the entire sequence of sort fields is common to both files, all are compared.

At least one pair of sort fields is required; field formats must be the same. In some cases, you can redefine a field's format using the DEFINE command. If the field names differ, use the AS phrase to rename the second sort field to match the first. When the AS phrase is used in a MATCH request, the specified field is automatically renamed in the resulting HOLD file.

When you are merging files with common sort fields, the following assumptions are made:

- If one of the sort fields is a subset of the other, a one-to-many relationship is assumed.
- If neither of the sort fields is a subset of the other, a one-to-one relationship is assumed. At most, one matching record is retrieved.

## Example

## MATCH Processing With Common High Order Sort Fields

To understand common high order sort fields more clearly, consider some of the data from the following data sources,

| EMPLOYEE data source |           | EDUCFILE data source |             |
|----------------------|-----------|----------------------|-------------|
| EMP_ID               | LAST_NAME | EMP_ID               | COURSE_CODE |
| 071382660            | STEVENS   | 071382660            | 101         |
| 119329144            | BANNING   | 21228911             | 103         |
| 112847612            | SMITH     | 112847612            | 103         |

and this MATCH request:

```
MATCH FILE EMPLOYEE
SUM LAST_NAME BY EMP_ID
RUN
FILE EDUCFILE
SUM COURSE_CODE BY EMP_ID
AFTER MATCH HOLD OLD-OR-NEW
END
```

MATCH processing occurs as follows:

- Since there is a common high-order sort field (EMP\_ID), the MATCH logic begins by matching the EMP\_ID values in the first records of EMPLOYEE and EDUCFILE.

- The first records match (each has the same EMP\_ID), so Record 1 is written to the HOLD file:

Record 1: 071382660 STEVENS 101

- The second records match (each has the value 112847612), so Record 2 is written to the HOLD file:

Record 2: 112847612 SMITH 103

- When the fifth records do not match (EMPLOYEE has the value 119329144 and EDUCFILE has the value 212289111), the record with the lower value is written to the HOLD file and a space is inserted for the missing value:

Record 5: 119329144 BANNING

- Similarly, the 21228911 record exists only in EDUCFILE, and is written as:

Record 8: 21228911 103

The following code produces a report of the records in the HOLD file:

```
TABLE FILE HOLD
PRINT *
END
```

The output is:

|           |           |             |
|-----------|-----------|-------------|
| PAGE 1    |           |             |
| EMP_ID    | LAST_NAME | COURSE_CODE |
| -----     | -----     | -----       |
| 071382660 | STEVENS   | 101         |
| 112847612 | SMITH     | 103         |
| 117593129 | JONES     | 203         |
| 119265415 | SMITH     | 108         |
| 119329144 | BANNING   |             |
| 123764317 | IRVING    |             |
| 126724188 | ROMANS    |             |
| 212289111 |           | 103         |
| 219984371 | MCCOY     |             |
| 315548712 |           | 108         |
| 326179357 | BLACKWOOD | 301         |
| 451123478 | MCKNIGHT  | 101         |
| 543729165 | GREENSPAN |             |
| 818692173 | CROSS     | 302         |

**Example****Merging With a Common High Order Sort Field**

This request combines data from the JOBFIL and PROD data sources. The sort fields are JOBCODE and PROD\_CODE, renamed as JOBCODE:

```
MATCH FILE JOBFIL
PRINT JOB_DESC
BY JOBCODE
RUN
FILE PROD
PRINT PROD_NAME
BY PROD_CODE AS 'JOBCODE'
AFTER MATCH HOLD OLD-OR-NEW
END
```

**Example****Merging Without a Common High Order Sort Field**

If there are no common high-order sort fields, a match is performed on a record-by-record basis. The following request matches the data and produces the HOLD file:

```
MATCH FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY EMP_ID
RUN
FILE EMPLOYEE
PRINT EMP_ID
BY LAST_NAME BY FIRST_NAME
AFTER MATCH HOLD OLD-OR-NEW
END
```

```
TABLE FILE HOLD
PRINT *
END
```

The retrieved records from the two data sources are written to the HOLD file; no values are compared.

The output is:

PAGE 1

| EMP_ID    | LAST_NAME | FIRST_NAME | LAST_NAME | FIRST_NAME | EMP_ID    |
|-----------|-----------|------------|-----------|------------|-----------|
| 071382660 | STEVENS   | ALFRED     | BANNING   | JOHN       | 119329144 |
| 112847612 | SMITH     | MARY       | BLACKWOOD | ROSEMARIE  | 326179357 |
| 117593129 | JONES     | DIANE      | CROSS     | BARBARA    | 818692173 |
| 119265415 | SMITH     | RICHARD    | GREENSPAN | MARY       | 543729165 |
| 119329144 | BANNING   | JOHN       | IRVING    | JOAN       | 123764317 |
| 123764317 | IRVING    | JOAN       | JONES     | DIANE      | 117593129 |
| 126724188 | ROMANS    | ANTHONY    | MCCOY     | JOHN       | 219984371 |
| 219984371 | MCCOY     | JOHN       | MCKNIGHT  | ROGER      | 451123478 |
| 326179357 | BLACKWOOD | ROSEMARIE  | ROMANS    | ANTHONY    | 126724188 |
| 451123478 | MCKNIGHT  | ROGER      | SMITH     | MARY       | 112847612 |
| 543729165 | GREENSPAN | MARY       | SMITH     | RICHARD    | 119265415 |
| 818692173 | CROSS     | BARBARA    | STEVENS   | ALFRED     | 071382660 |



# Fine Tuning MATCH Processing

You can fine tune the MATCH process using the PRINT and SUM commands. To understand their difference, you should have an understanding of the one-to-many relationship: SUM generates one record from many, while PRINT displays each individual record. Through proper choices of BY fields, it is possible to use only the SUM command and get the same result that PRINT would produce.

## Example

### Using Display Commands in MATCH Processing

To best illustrate the effects of PRINT and SUM on the MATCH process, consider data sources A and B and the series of requests that follow:

| A  |    |     | B  |    |    |
|----|----|-----|----|----|----|
| F1 | F2 | F3  | F1 | F4 | F5 |
| 1  | x  | 100 | 1  | a  | 10 |
| 2  | y  | 200 | 1  | b  | 20 |
|    |    |     | 2  | c  | 30 |
|    |    |     | 2  | d  | 40 |

**Request 1:** This request sums the fields F2 and F3 from file A, sums the fields F4 and F5 from file B, and uses F1 as the common high order sort field.

```
MATCH FILE A
SUM F2 AND F3 BY F1
RUN
FILE B
SUM F4 AND F5 BY F1
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains the following data:

| F1 | F2 | F3  | F4 | F5 |
|----|----|-----|----|----|
| 1  | x  | 100 | b  | 30 |
| 2  | y  | 200 | d  | 70 |

Note that the resulting file contains only 1 record for each common high order sort field.

**Request 2:** This request sums fields F2 and F3 from file A, prints fields F4 and F5 from file B, and uses F1 as the common high order sort field.

```
MATCH FILE A
SUM F2 AND F3 BY F1
RUN
FILE B PRINT F4 AND F5 BY F1
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains:

| F1 | F2 | F3  | F4 | F5 |
|----|----|-----|----|----|
| 1  | x  | 100 | a  | 10 |
| 1  | x  | 100 | b  | 20 |
| 2  | y  | 200 | c  | 30 |
| 2  | y  | 200 | d  | 40 |

Note that the records from file A are duplicated for each record from file B.

**Request 3:** This request prints fields F2 and F3 from file A, sums fields F4 and F5 from file B, and uses F1 as the common high order sort field.

```
MATCH FILE A
PRINT F2 AND F3 BY F1
RUN
FILE B
SUM F4 AND F5 BY F1
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains:

| F1 | F2 | F3  | F4 | F5 |
|----|----|-----|----|----|
| 1  | x  | 100 | b  | 30 |
| 2  | y  | 200 | d  | 70 |

Note that each record from file A is included, but only the last record from file B for each common high order sort field.

**Request 4:** This request prints fields F2 and F3 from file A, prints fields F4 and F5 from file B, and uses F1 as the common high order sort field.

```
MATCH FILE A
PRINT F2 AND F3 BY F1
RUN
FILE B PRINT F4 AND F5 BY F1
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains:

| F1 | F2 | F3  | F4 | F5 |
|----|----|-----|----|----|
| 1  | x  | 100 | a  | 10 |
| 1  |    | 0   | b  | 20 |
| 2  | y  | 200 | c  | 30 |
| 2  |    | 0   | d  | 40 |

Note the blank value for F2 and the 0 for F3.

**Request 5:** This request sums the fields F2 and F3 from file A, sums the field F5 from file B and sorts it by field F1, the common high order sort field, and by F4.

```
MATCH FILE A
SUM F2 AND F3 BY F1
RUN
FILE B
SUM F5 BY F1 BY F4
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains:

| F1 | F2 | F3  | F4 | F5 |
|----|----|-----|----|----|
| 1  | x  | 100 | a  | 10 |
| 1  | x  | 100 | b  | 20 |
| 2  | y  | 200 | c  | 30 |
| 2  | y  | 200 | d  | 40 |

Note that the records for file A are printed for every occurrence of the record in file B.

# Universal Concatenation

With Universal Concatenation, you can retrieve data from unlike data source types in a single request; all data, regardless of source, appears to come from a single file. The MORE phrase can concatenate all types of data sources (such as, FOCUS, DB2, IMS, VSAM), provided they share corresponding fields with the same format. You can use WHERE and IF selection tests in conjunction with MORE. For related information, see Chapter 5, *Selecting Records for Your Report*.

To use MORE, you must divide your request into:

- One main request that retrieves the first data source and defines the data fields, sorting criteria, and output format for all data.
- Subrequests that define the data sources and fields to be concatenated to the data of the main request. The fields printed and sorted by the main request must exist in each concatenated data source. If they do not, you must create them as virtual fields.

During retrieval, data is gathered from each data source in turn, then all data is sorted and the output formatted as specified in the main request.

## Syntax

### How to Concatenate Data Sources

The MORE phrase, which is accessible within the TABLE and MATCH commands, specifies how to concatenate data from sources with dissimilar Master Files.

The syntax for a request that concatenates data sources is

```
{TABLE|MATCH} FILE file1

 main request
MORE
FILE file2
 subrequest
MORE
FILE file3
 subrequest
MORE
.
.
.
{END|RUN}
```

where:

**TABLE|MATCH**

Begins the request that concatenates the data sources.

*file1*

Is the name of the first data source.

*main request*

Is a request, without END or RUN, that retrieves the first data source and defines the data fields, sorting criteria, and output format for all data. WHERE and IF criteria in the main request apply only to *file1*.

When concatenating files within the TABLE command you can also define calculated values for the first data source.

**MORE**

Begins a subrequest. There is no limit to the number of subrequests (other than available memory).

**FILE *file2***

Defines *file2* as the second data source for concatenation.

*subrequest*

Is a subrequest. Subrequests can only include WHERE and IF phrases.

**END|RUN**

Ends the request.

## Example

## Concatenating Data Sources

Both the EMPLOYEE and the EXPERSON data sources contain employee information. You can concatenate their common data into a single file:

- EMPLOYEE contains the field values EMP\_ID=123456789 and CURR\_SAL=50.00.
- EXPERSON contains the field values SSN=987654321 and WAGE=100.00.

The following annotated request concatenates the two data sources:

```
DEFINE FILE EXPERSON
1. EMP_ID/A9 = SSN;
 CURR_SAL/D12.2 = WAGE;
 END
2. TABLE FILE EMPLOYEE
 PRINT CURR_SAL
 BY EMP_ID
3. MORE
 FILE EXPERSON
 END
```

1. The request must re-map the field names and formats in the EXPERSON data source to match those used in the main request.
2. The main request names the first data source in the concatenation, EMPLOYEE. It also defines the print and sort fields for both data sources.
3. The MORE phrase starts the subrequest that concatenates the next data source, EXPERSON. No display commands are allowed in the subrequest. IF and WHERE criteria are the only report components permitted in a subrequest.

## Field Name and Format Matching

All fields referenced in the main request must either exist with the same names and formats in all the concatenated files, or be re-mapped to those names and formats using virtual fields. Referenced fields include those used in COMPUTE commands, headings, aggregation phrases, sort phrases, and the PRINT, LIST, SUM, COUNT, WRITE, or ADD commands.

A successful format match means that:

| Usage Format Type | Correspondence                                                                   |
|-------------------|----------------------------------------------------------------------------------|
| A                 | Format type and length must be equal.                                            |
| I, F, D           | Format type must be the same.                                                    |
| P                 | Format type and scale must be equal.                                             |
| DATE (new)        | Format information (type, length, components, and order) must always correspond. |
| DATE (old)        | Edit options must be the same.                                                   |
| DATE -TIME        | Format information (type, length, components, and order) must always correspond. |

Text (TX) fields and CLOB fields (if supported) cannot be concatenated.

## Example

## Matching Field Names and Formats

The following annotated example concatenates data from the EMPDATA and PAYHIST data sources. Appendix A, *Master Files and Diagrams*, contains the Master Files referenced in the request.

### Tip:

PAYHIST is a fixed format file. You will need to issue a FILEDEF command in order to use it. See the *Developing Applications* manual for information on the FILEDEF command.

```
DEFINE FILE EMPDATA
1. NEWID/A11 = EDIT (ID, '999-99-9999');
 END

 DEFINE FILE PAYHIST
1. NEWID/A11 = EDIT (SSN, '999-99-9999');
 CSAL/D12.2M = NEW_SAL;
 END

2. TABLE FILE EMPDATA
 HEADING
 "EMPLOYEE SALARIES"
 " "
3. PRINT CSAL
3. BY NEWID AS 'EMPLOYEE ID'
4. WHERE CSAL GT 65000

5. MORE
 FILE PAYHIST
6. WHERE NEW_SAL GT 500
 END
```

1. Defines NEWID with the same name and format as the sort field referenced in the main request.
2. The main request contains all formatting for the resulting report and names the first data source to be concatenated.
3. The main request also contains all printing and sorting information. The fields printed and the sort fields must exist as real or virtual fields in each data source.
4. The WHERE criterion in the main request applies only to the EMPDATA data source.
5. The MORE phrase concatenates the PAYHIST data source to the EMPDATA data source.
6. This WHERE criterion applies only to the PAYHIST data source. Notice that it references a field that is not defined in the EMPDATA data source.

In the resulting report, the EMPLOYEE ID values that start with 000 are from EMPDATA and the values that start with 100 are from PAYHIST:

```
PAGE 1

EMPLOYEE SALARIES

EMPLOYEE ID SALARY

000-00-0030 $70,000.00
000-00-0070 $83,000.00
000-00-0200 $115,000.00
000-00-0230 $80,500.00
000-00-0300 $79,000.00
100-10-1689 $842.90
 $982.90
100-11-9950 $508.75
100-14-2166 $876.45
100-15-5843 $508.75
100-16-2791 $567.89
100-16-4984 $1,236.78
100-17-5025 $734.56
100-18-9299 $567.89
```

## Merging Concatenated Data Sources

You can use the MORE phrase in a MATCH request to merge up to six sets of concatenated data sources.

You must meet all MATCH requirements in the main request. All data sources to be merged must be sorted by at least one field with a common format.

The MATCH request results in a HOLD file containing the merged data. You can specify how you want each successive file merged using an AFTER MATCH command. For example, you can retain:

- All records from both files (OLD-OR-NEW). This is the default.
- Only records common to both files (OLD-AND-NEW).
- Records from the first file with no match in the second file (OLD-NOT-NEW).
- Records from the second file with no match in the first file (NEW-NOT-OLD).
- All non-matching records from both files; that is, records that were in either one of the files but not both (OLD-NOR-NEW).
- All records from the first file with all matching records from the second file (OLD).
- All records from the second file with all matching records from the first file (NEW).



## Syntax

## How to Merge Concatenated Data Sources

The syntax for a MATCH request against concatenated data sources is:

```
1. MATCH FILE file1
 main request
 MORE
2. FILE file2
 subrequest
 MORE
3. FILE file3
 subrequest
 RUN
4. FILE file4
 main request
5. [AFTER MATCH merge_phrase]
 MORE
6. FILE file5
 subrequest
 MORE
7. FILE file6
 subrequest
 RUN
8. FILE file7
 main request
9. [AFTER MATCH merge_phrase]
 MORE
10. FILE file8
 subrequest
 MORE
11. FILE file9
 subrequest
 END
```

1. Starts the first answer set in the MATCH. *File1* is the first data source in the first answer set.
2. Concatenates *file2* to *file1* in the first MATCH answer set.
3. Concatenates *file3* to *file1* and *file2* in the first MATCH answer set.
4. Starts the second answer set in the MATCH. *File4* is the first data source in the second answer set.
5. All data concatenated in the first answer set is merged with the data concatenated in the second answer set using the AFTER MATCH merge\_phrase in the second answer set.
6. Concatenates *file5* to *file4* in the second MATCH answer set.
7. Concatenates *file6* to *file4* and *file5* in the second MATCH answer set.
8. Starts the third answer set in the MATCH. *File7* is the first data source in the third answer set.
9. All merged data from the first and second answer sets, now a HOLD file, is merged with the data concatenated in the third answer set using the AFTER MATCH merge\_phrase in the third answer set. This final set of merged data is stored in a HOLD file.
10. Concatenates *file8* to *file7* in the third MATCH answer set.
11. Concatenates *file9* to *file7* and *file8* in the third MATCH answer set.

## Using Sort Fields in MATCH Requests

If the data sources in the MATCH share common high-order sort fields with identical names and formats, the MATCH process merges records with matching sort field values from each of the files. If the two data sources in the MATCH have the same sort field with different names, you can change one of the names with an AS phrase.

If the files in the MATCH do not share a high-order sort field, the fields are not compared. Instead, the fields from the first record in each data source are merged to create the first record in the HOLD file, and so on for all remaining records.

### Example

### Merging Concatenated Data Sources With Common High Order Sort Fields

The following annotated sample stored procedure illustrates MATCH with MORE, using a common sort field:

```
1. DEFINE FILE EMPDATA
 CURR_SAL/D12.2M = CSAL;
 FIRST_NAME/A10 = FN;
 EID/A9 = PIN;
 END

 -*Start MATCH.

2. MATCH FILE EMPLOYEE
 SUM CURR_SAL AS 'CURRENT'
 FIRST_NAME AS 'FIRST'
 BY EID AS 'SSN'
 -*Concatenate file EMPDATA to EMPLOYEE to form first MATCH answer set.
3. MORE
 FILE EMPDATA
 RUN
 -*Second MATCH answer set:

4. FILE TRAINING
 PRINT EXPENSES
5. BY PIN AS 'SSN'
6. AFTER MATCH HOLD OLD-OR-NEW
 END

 -*Print merged file:

7. TABLE FILE HOLD
 PRINT *
 END
```

1. Defines the EMPDATA fields needed for concatenating it to EMPLOYEE.
2. Starts the MATCH and the main request in the concatenation. The main request defines all printing and sorting for the concatenated files. The sort field is called SSN in the resulting file.
3. Concatenates file EMPDATA to EMPLOYEE. This concatenated file becomes the OLD file in the match.
4. Creates the NEW file in the match.
5. Uses an AS phrase to change the name of the sort field in the NEW file to the same name as the sort field in the OLD file.
6. Defines the merge procedure. All records from the NEW file, the OLD file, and both files are included in the final HOLD file.
7. Prints the values from the merged file.

The output is:

PAGE 1

| SSN       | CURRENT     | FIRST    | EXPENSES |
|-----------|-------------|----------|----------|
| ---       | -----       | -----    | -----    |
| 000000010 | \$55,500.00 | DANIEL   | 2,300.00 |
| 000000020 | \$62,500.00 | MICHAEL  | .        |
| 000000030 | \$70,000.00 | LOIS     | 2,600.00 |
| 000000030 | \$70,000.00 | LOIS     | 2,300.00 |
| 000000040 | \$62,500.00 | RUTH     | 3,400.00 |
| 000000050 | \$54,100.00 | PETER    | 3,300.00 |
| 000000060 | \$55,500.00 | DORINA   | .        |
| 000000070 | \$83,000.00 | EVELYN   | .        |
| 000000080 | \$43,400.00 | PAMELA   | 3,200.00 |
| 000000080 | \$43,400.00 | PAMELA   | 3,350.00 |
| 000000090 | \$33,000.00 | MARIANNE | .        |
| 000000100 | \$32,400.00 | TIM      | 3,100.00 |
| 000000110 | \$19,300.00 | ANTHONY  | 1,800.00 |
| 000000110 | \$19,300.00 | ANTHONY  | 2,500.00 |
| 000000110 | \$19,300.00 | ANTHONY  | 2,400.00 |
| 000000120 | \$49,500.00 | KATE     | 2,200.00 |
| 000000130 | \$62,500.00 | MARCUS   | .        |

## Example

### Merging Concatenated Data Sources Without a Common Sort Field

In this example, the merged data sources do not share a sort field:

```
DEFINE FILE EMPDATA
CURR_SAL/D12.2M = CSAL;
FIRST_NAME/A10 = FN;
EID/A9 = PIN;
END

-*Start MATCH

MATCH FILE EMPLOYEE
SUM CURR_SAL AS 'CURRENT'
 FIRST_NAME AS 'FIRST'
BY EID AS 'SSN'

-*Concatenate EMPDATA to EMPLOYEE to form the first MATCH answer set

MORE
FILE EMPDATA
RUN

-*Second MATCH answer set:

FILE TRAINING
PRINT EXPENSES
BY PIN AS 'EID'
AFTER MATCH HOLD OLD-OR-NEW
END

-*Print merged file:

TABLE FILE HOLD
PRINT *
END
```

The AS phrase changes the answer set. Since the sort fields no longer have the same names, the fields are merged with no regard to matching records.

The output is:

PAGE 1

| SSN       | CURRENT     | FIRST    | EID       | EXPENSES |
|-----------|-------------|----------|-----------|----------|
| ---       | -----       | -----    | ---       | -----    |
| 000000010 | \$55,500.00 | DANIEL   | 000000010 | 2,300.00 |
| 000000020 | \$62,500.00 | MICHAEL  | 000000030 | 2,600.00 |
| 000000030 | \$70,000.00 | LOIS     | 000000030 | 2,300.00 |
| 000000040 | \$62,500.00 | RUTH     | 000000040 | 3,400.00 |
| 000000050 | \$54,100.00 | PETER    | 000000050 | 3,300.00 |
| 000000060 | \$55,500.00 | DORINA   | 000000080 | 3,200.00 |
| 000000070 | \$83,000.00 | EVELYN   | 000000080 | 3,350.00 |
| 000000080 | \$43,400.00 | PAMELA   | 000000100 | 3,100.00 |
| 000000090 | \$33,000.00 | MARIANNE | 000000110 | 1,800.00 |
| 000000100 | \$32,400.00 | TIM      | 000000110 | 2,500.00 |
| 000000110 | \$19,300.00 | ANTHONY  | 000000110 | 2,400.00 |
| 000000120 | \$49,500.00 | KATE     | 000000120 | 2,200.00 |
| 000000130 | \$62,500.00 | MARCUS   | 000000140 | 3,600.00 |
| 000000140 | \$62,500.00 | VERONICA | 000000150 | 3,400.00 |
| 000000150 | \$40,900.00 | KARL     | 000000160 | 1,000.00 |
| 000000160 | \$62,500.00 | ROSE     | 000000180 | 1,250.00 |
| 000000170 | \$30,800.00 | WILLIAM  | 000000190 | 3,150.00 |

## Cartesian Product

Cartesian product enables you to generate a report containing all combinations of non-related records or data instances in a multi-path request. This means that if a parent segment has three child instances on one path and two child instances on another path, when CARTESIAN is ON a request that references the parent segment and both children will generate six records. When CARTESIAN is OFF, the same request will generate only three records.

For related information about controlling how selection tests are applied to child segments on independent paths, see Chapter 5, *Selecting Records for Your Report*.

### Syntax

#### How to Enable/Disable Cartesian Product

SET CARTESIAN = {[OFF](#)|ON}

where:

[OFF](#)

Disables Cartesian product. OFF is the default setting.

[ON](#)

Enables Cartesian product and generates all possible combinations of non-related records.

SET CARTESIAN may also be issued within a request.

Reference

Usage Notes for Cartesian Product

- Cartesian product is performed on the lowest segment common to all paths, whether or not a field in that segment is referenced.
- Short paths do not display in requests with Cartesian product.
- The SET CARTESIAN command is disabled when ACROSS is specified, and a warning message is issued.
- The SUM display command and the TOT. prefix operator have no effect on Cartesian product.
- SUM, COMPUTE, and WITHIN in combination with the PRINT display command are performed on the Cartesian product.
- ON TABLE COLUMN-TOTAL is automatically generated on the Cartesian product.
- NOSPLIT is disabled if specified in combination with the SET CARTESIAN command, and no warning message is issued.
- MATCH is not supported with the SET CARTESIAN command. A warning message is not issued if MATCH is requested, and the request is processed as if CARTESIAN is set to OFF.
- TABLEF is not supported with the SET CARTESIAN command.

Example

Reporting With Cartesian Product

When CARTESIAN is set to ON, the following multi-path request produces a report containing all possible combinations of models and standards for each car:

```
SET CARTESIAN=ON
TABLE FILE CAR
PRINT MODEL STANDARD
BY CAR
IF CAR EQ 'JAGUAR'
END
```

The output is:

PAGE 1

| CAR    | MODEL       | STANDARD                   |
|--------|-------------|----------------------------|
| ---    | -----       | -----                      |
| JAGUAR | V12XKE AUTO | POWER STEERING             |
|        | V12XKE AUTO | RECLINING BUCKET SEATS     |
|        | V12XKE AUTO | WHITEWALL RADIAL PLY TIRES |
|        | V12XKE AUTO | WRAP AROUND BUMPERS        |
|        | V12XKE AUTO | 4 WHEEL DISC BRAKES        |
|        | XJ12L AUTO  | POWER STEERING             |
|        | XJ12L AUTO  | RECLINING BUCKET SEATS     |
|        | XJ12L AUTO  | WHITEWALL RADIAL PLY TIRES |
|        | XJ12L AUTO  | WRAP AROUND BUMPERS        |
|        | XJ12L AUTO  | 4 WHEEL DISC BRAKES        |

When CARTESIAN is set to OFF (the default), the same request results in a report from the CAR data source containing a list of models and standards without logical relationships.

The output is:

PAGE 1

| CAR    | MODEL       | STANDARD                   |
|--------|-------------|----------------------------|
| ---    | -----       | -----                      |
| JAGUAR | V12XKE AUTO | POWER STEERING             |
|        | XJ12L AUTO  | RECLINING BUCKET SEATS     |
|        | .           | WHITEWALL RADIAL PLY TIRES |
|        | .           | WRAP AROUND BUMPERS        |
|        | .           | 4 WHEEL DISC BRAKES        |



---

## CHAPTER 15

# Improving Report Processing

### Topics:

- Rotating a Data Structure for Enhanced Retrieval
- Optimizing Retrieval Speed for FOCUS Data Sources
- Automatic Indexed Retrieval
- Data Retrieval Using TABLEF
- Preserving the Internal Matrix of Your Last Report

The following high-performance methods are supported to optimize data retrieval and report processing:

- Temporary rotation of network and hierarchical data sources to create an alternate view of the data, from which you can report. See *Rotating a Data Structure for Enhanced Retrieval* on page 15-2.
- Automatic alternate file views with the AUTOPATH feature. See *Optimizing Retrieval Speed for FOCUS Data Sources* on page 15-4.
- Automatic indexed retrieval (AUTOINDEX), which takes advantage of indexed fields used in equality or range tests. See *Automatic Indexed Retrieval* on page 15-4.
- Retrieval of pre-sorted data using the TABLEF command. See *Data Retrieval Using TABLEF* on page 15-6.
- Preserving a report's internal matrix using the SAVEMATRIX parameter. See *Preserving the Internal Matrix of Your Last Report* on page 15-7.

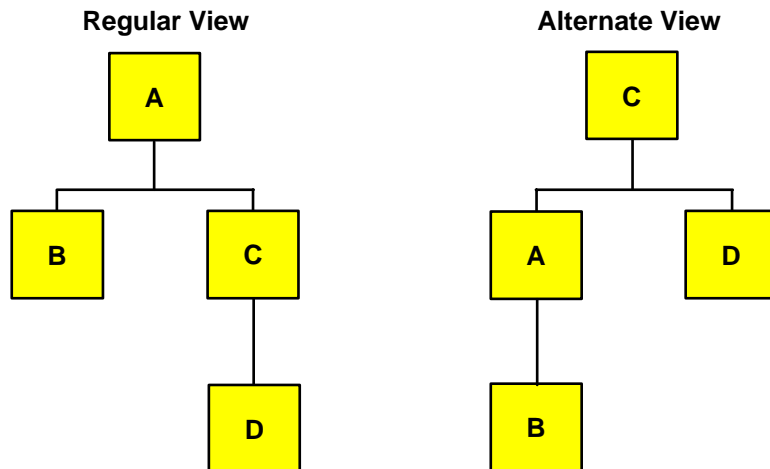
**Note:** These techniques may not be available for the data source you are working with. See your data adapter documentation to determine if a technique is valid for your data source.

## Rotating a Data Structure for Enhanced Retrieval

If you are using certain network or hierarchical data sources such as IMS, CA-IDMS/DB, or FOCUS, you can rotate the data source, creating an alternate view which changes some of the segment relationships and enables you to access the segments in a different order. By reporting from an alternate view, you can do the following:

- Change the access path. For example, you can access data in a lower segment more quickly by promoting that segment to a higher level.
- Change the path structure of a data source. This option is especially helpful if you wish to create a report using several sort fields that are on different paths in the file. By changing the view of the file hierarchy, all the desired sort fields can be on the same path.

For example, consider the regular and alternate views below:



Since C is the root segment in the alternate view, particular instances of C can be selected faster.

### Syntax

#### How to Request an Alternate View

To request an alternate view, add the name of a field—one found in the alternate root segment—to the file name in the TABLE command, separated by a period (.):

```
TABLE FILE filename.fieldname
```

## Reference

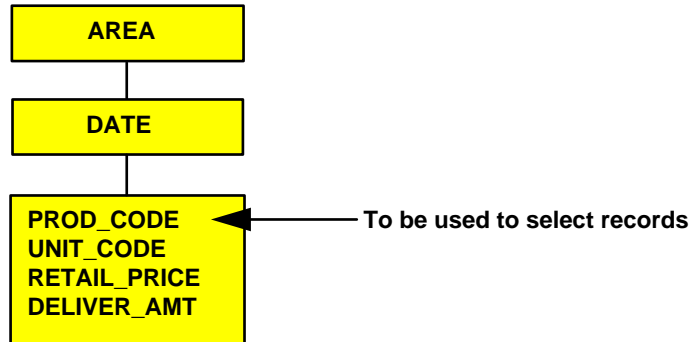
### Usage Notes for Restructuring Data

- If you use a non-indexed field, each segment instance is retrieved until the specified record is found. This process is, therefore, less efficient than using an indexed field.
- When you use the alternate view feature on a particular child segment, the data retrieved from that segment will be retrieved in physical order, not logical order. This is so because the child becomes a root segment for the report request, and there are no logical pointers between the child segments of different parents.
- Alternate view on an indexed field is a special case that uses the index for retrieval. When you perform an alternate view on an indexed field, you enhance the speed of retrieval. However, you must include an equality test on the indexed field, for example WHERE (MONTH EQ 1) OR (MONTH EQ 2), in order to benefit from the performance improvement.
- A field name specified in an alternate file view may not be qualified or exceed 12 characters.
- Automatic Indexed Retrieval (AUTOINDEX) is never invoked in a TABLE request against an alternate file view.

## Example

### Restructuring Data

Consider the following data structure, in which PROD\_CODE is an indexed field:



You could issue the following request to promote the segment containing PROD\_CODE to the top of the hierarchy, thereby enabling quicker access to the data in that segment.

```

TABLE FILE SALES.PROD_CODE
"SALES OF B10 DISTRIBUTED BY AREA"
SUM UNIT_SOLD AND RETAIL_PRICE
BY AREA
WHERE PROD_CODE EQ 'B10'
ON TABLE COLUMN-TOTAL
END

```

## Optimizing Retrieval Speed for FOCUS Data Sources

When the AUTOPATH parameter is set ON, an optimized retrieval path—that is, one in which the lowest retrieved segment is the entry point—is selected dynamically. It is equivalent to the alternate view syntax

`TABLE FILE filename.fieldname`

where:

`fieldname`

Is not indexed. Retrieval starts at the segment in which *fieldname* resides.

The system determines whether optimized retrieval is appropriate by analyzing the fields referenced in a request and the data source structure. For more information on the AUTOPATH parameter, see the *Developing Applications* manual.

### Tip:

Another way to optimize data retrieval is by using intelligent partitioning in requests that do not require retrieval from every partition. For information on efficiency considerations for FOCUS data sources, including intelligent partitioning, see the *Describing Data* manual.

## Automatic Indexed Retrieval

Automatic indexed retrieval (AUTOINDEX) optimizes the speed of data retrieval in FOCUS and Fusion data sources. To take advantage of automatic indexed retrieval, a TABLE request must contain an equality or range test on an indexed field in the highest segment referenced in the request.

This method is not supported if a:

- Range test applies to a packed data value.
- Request specifies an alternate view (that is, TABLE FILE *filename.fieldname*).
- Request contains the code BY HIGHEST or BY LOWEST.

For related information on AUTOINDEX, see the *Developing Applications* manual.

### Syntax

#### How to Use Indexed Retrieval

`SET AUTOINDEX = {ON|OFF}`

where:

ON

Uses indexed data retrieval for optimized speed when possible. The request must contain an equality or range test on an indexed field in the highest segment referenced in the request.

OFF

Uses sequential data retrieval unless a request specifies an indexed view (TABLE FILE *filename.indexed\_fieldname*) and contains an equality test on *indexed\_fieldname*. In that case, indexed data retrieval is automatically performed. This value is the default; however, the default may have been changed in a supported profile. You can check your setting by issuing the ? SET command.

## Example

## Using Indexed Retrieval

The following Master File is referenced in the examples that follow:

```

FILENAME=SALES,SUFFIX=FOC,
 SEGNAME=STOR_SEG,SEGTYPE=S1,
 FIELDNAME=AREA,ALIAS=LOC,FORMAT=A1,$
 SEGNAME=DATE_SEG,PARENT=STOR_SEG,SEGTYPE=SH1,
 FIELDNAME=DATE,ALIAS=DTE,FORMAT=A4MD,$
 SEGNAME=DEPT,PARENT=DATE_SEG,SEGTYPE=S1,
 FIELDNAME=DEPARTMENT,ALIAS=DEPT,FORMAT=A5,FIELDTYPE=I,$
 FIELDNAME=DEPT_CODE,ALIAS=DCODE,FORMAT=A3,FIELDTYPE=I,$
 FIELDNAME=PROD_TYPE,ALIAS=PTYPE,FORMAT=A10,FIELDTYPE=I,$
 SEGNAME=INVENTORY,PARENT=DEPT,SEGTYPE=S1,$
 FIELDNAME=PROD_CODE,ALIAS=PCODE,FORMAT=A3,FIELDTYPE=I,$
 FIELDNAME=UNIT_SOLD,ALIAS=SOLD,FORMAT=I5,$
 FIELDNAME=RETAIL_PRICE,ALIAS=RP,FORMAT=D5.2M,$
 FIELDNAME=DELIVER_AMT,ALIAS=SHIP,FORMAT=I5,$

```

The following procedure contains an equality test on DEPT\_CODE and PROD\_CODE. DEPT\_CODE is used for indexed retrieval since it is in the higher of the referenced segments.

```

SET AUTOINDEX=ON
TABLE FILE SALES
SUM UNIT_SOLD RETAIL_PRICE
IF DEPT_CODE EQ 'H01'
IF PROD_CODE EQ 'B10'
END

```

If your TABLE request contains an equality or range test against more than one indexed field in the same segment, AUTOINDEX uses the first index referenced in that segment for retrieval. The following stored procedure contains an equality test against two indexed fields. Since DEPT\_CODE appears before PROD\_TYPE in the Master File, AUTOINDEX uses DEPT\_CODE for retrieval.

```

SET AUTOINDEX=ON
TABLE FILE SALES
SUM UNIT_SOLD AND RETAIL_PRICE
IF PROD_TYPE EQ 'STEREO'
IF DEPT_CODE EQ 'H01'
END

```

Indexed retrieval is not invoked if the equality or range test is run against an indexed field that does not reside in the highest referenced segment. In the following example, indexed retrieval is not performed because the request contains a reference to AREA, a field in the STOR\_SEG segment:

```

SET AUTOINDEX=ON
TABLE FILE SALES
SUM UNIT_SOLD AND RETAIL_PRICE
BY AREA
IF PROD_CODE EQ 'B10'
IF PROD_TYPE EQ 'STEREO'
END

```

## Data Retrieval Using TABLEF

TABLEF is a variation of the TABLE command that provides a fast method of retrieving data that is already stored in the order required for printing and no additional sorting is required

Using TABLEF, records are retrieved in the logical sequence from the data source. The standard report request syntax applies, subject to the following rules:

- Any BY phrases must be compatible with the logical sequence of the data source. BY phrases are used only to establish control breaks, not to change the order of the records.
- ACROSS phrases are not permitted.
- Multiple display commands are not permitted. Only one display command may be used.
- After the report is executed, RETYPE, HOLD, and SAVE are not available. However, you can produce an extract file if you include ON TABLE HOLD or ON TABLE SAVE as part of the request.
- NOSPLIT is not compatible with the TABLEF command and produces a FOC037 error message.
- TABLEF can be used with HOLD files and other non-FOCUS data sources when the natural sort sequence of both the request and the data are the same.
- TABLEF is not supported with SET EMPTYREPORT. When a TABLEF request retrieves zero records, EMPTYREPORT behaves as if EMPTYREPORT were set to ON.
- The DST. prefix operator is not permitted.

### Example

#### Printing Using Fast Table Retrieval

If you previously created a HOLD file from the EMPLOYEE data source, sorted by the CURR\_SAL, LAST\_NAME, and FIRST\_NAME fields, you can issue the following TABLEF request:

```
TABLEF FILE HOLD
PRINT CURR_SAL AND LAST_NAME AND FIRST_NAME
END
```

# Preserving the Internal Matrix of Your Last Report

An internal matrix is generated with each TABLE, FML, GRAPH, and MATCH request. These requests are available for the duration of your session, or until you generate a new report or graph that overwrites it.

While a report (or graph) request is available, you can:

- Extract and save data from it using the HOLD, SAVE, and SAVB commands.
- Redisplay it using the RETYPE or REPLOT commands.

If you wish to save the matrix from your last request to protect it from being over-written when using Dialogue Manager commands, you can activate the SET SAVEMATRIX feature.

**Note:** SET SAVEMATRIX is not available with the TABLEF command.

## Syntax

### How to Save an Internal Matrix

```
SET SAVEMATRIX = {ON|OFF}
```

where:

ON

Saves the internal matrix from the last report request, preventing it from being overwritten.

OFF

Overwrites the internal matrix for each request. This value is the default.

## Example

### Saving a Report's Internal Matrix

The following request creates a report, then executes a procedure that contains a Dialogue Manager command (which would otherwise overwrite the internal matrix), and recalls the report using the RETYPE command:

```
SET SAVEMATRIX = ON
TABLE FILE EMPLOYEE
.
.
.
END
EX DMFEX
RETYPE
```

---

## CHAPTER 16

# Creating Financial Reports

### Topics:

- Reporting With FML
- Creating Rows From Data
- Performing Inter-Row Calculations
- Referring to Rows
- Referring to Columns
- Referring to Cells
- Using Subroutines in Calculations
- Supplying Data Directly in the FML Request
- Inserting Rows of Free Text
- Adding Columns to an FML Report
- Creating Recursive Models
- Formatting an FML Report
- Suppressing Tagged Rows
- Saving and Retrieving Intermediate Report Results
- Creating HOLD Files From FML Reports

The Financial Modeling Language (FML) is designed for the special needs associated with creating, calculating, and presenting financially oriented data such as balance sheets, consolidations, or budgets. These reports are distinguished from other reports because calculations are inter-row as well as inter-column and each row or line represents a unique entry or series of entries that can be aggregated directly from the input data or calculated as some function of the data.



## Reporting With FML

FML is an integrated extension of the TABLE command. By adding the FOR phrase and the RECAP command, you can handle a vastly expanded range of applications.

Used in conjunction with Dialogue Manager, FML can be used to perform “what if” scenarios and develop complete decision support systems. These systems can take advantage of business intelligence features, such as statistical analysis and graphics, in addition to standard financial statements.

Procedures using FML are not hard-wired to the data. As in any other report request, these procedures can easily be changed. FML includes the following facilities:

- Row/column formatting: You can easily specify results in a row-by-row, column-by-column fashion (see *Performing Inter-Row Calculations* on page 16-12).
- Intermediate results: You can post FML results to an external file and pick them up at a later time for analysis. This is useful when intermediate results are developed and a final procedure consolidates the results later (see *Saving and Retrieving Intermediate Report Results* on page 16-32).
- Inline data entry: FML enables you to specify constants from within the procedure in addition to the data values retrieved from your data source (see *Supplying Data Directly in the FML Request* on page 16-24).
- Recursive reports: You can produce reports where the results from the end of one time period or column become the starting balance in the next. For example, you could use recursive reports to produce a cash flow projection (see *Creating Recursive Models* on page 16-28).

**Example****Sample FML Request**

The following annotated example illustrates several of the points discussed in *Reporting With FML* on page 16-2. Notice the similarity of this example to a typical reporting request, except for the addition of the two FML phrases. The example produces a simple asset sheet, contrasting the results of two years. Though the example is relatively simple and the results brief, an FML report could be quite comprehensive, using Dialogue Manager to create a flexible, interactive model.

```
TABLE FILE FINANCE
HEADING CENTER
"COMPARATIVE ASSET SHEET </2"
SUM AMOUNT ACROSS HIGHEST YEAR
WHERE YEAR EQ '1983' OR '1982'
1. FOR ACCOUNT
2. 1000 AS 'UTILITY PLANT' LABEL UTP OVER
2. 1010 TO 1050 AS 'LESS ACCUMULATED DEPRECIATION' LABEL UTPAD OVER
3. BAR OVER
4. RECAP UTPNET=UTP-UTPAD; AS 'TOTAL PLANT-NET' OVER
 BAR OVER
 2000 TO 3999 AS 'INVESTMENTS' LABEL INV OVER
5. "CURRENT ASSETS" OVER
 4000 AS 'CASH' LABEL CASH OVER
 5000 TO 5999 AS 'ACCOUNTS RECEIVABLE-NET' LABEL ACR OVER
 6000 AS 'INTEREST RECEIVABLE' LABEL ACI OVER
 6500 AS 'FUEL INVENTORY' LABEL FUEL OVER
 6600 AS 'MATERIALS AND SUPPLIES' LABEL MAT OVER
 6900 AS 'OTHER' LABEL MISC OVER
 BAR OVER
 RECAP TOTCAS = CASH+ACR+ACI+FUEL+MAT+MISC ; AS 'TOTAL CURRENT ASSETS' OVER
 BAR OVER
 7000 AS 'DEFERRED DEBITS' LABEL DEFDB OVER
 BAR OVER
6. RECAP TOTAL=UTPNET+INV+TOTCAS+DEFDB; AS 'TOTAL ASSETS' OVER
 BAR AS '='
 FOOTING CENTER
 "</2 *** PRELIMINARY ASSET SHEET BASED ON UNAUDITED FIGURES ***"
END
```

1. FOR and OVER are FML phrases that enable you to structure the report on a row-by-row basis.
2. LABEL assigns a variable name to a row item for use in a RECAP calculation. 1000 and 1010 TO 1050 are tags that identify the data values of the FOR field, ACCOUNT. A report row can be associated with a tag that represents a single data value (like 1000), multiple data values, or a range of values (like 1010 TO 1050).
3. BAR enables you to underline a column of numbers before performing a RECAP calculation.
4. The RECAP command creates a new value based on values already identified in the report with LABEL. In this case, the value UTPNET is derived from UTP and UTPAD and is renamed TOTAL PLANT-NET with an AS phrase to provide it with greater meaning on the report.
5. Like underlines, free text can be incorporated at any point in an FML report.
6. Notice that this RECAP command derives a total (TOTAL ASSETS) from values retrieved directly from the data source and from values derived from previous RECAP computations (UTPNET and TOTCAS).

The output is:

| COMPARATIVE ASSET SHEET       |             |             |
|-------------------------------|-------------|-------------|
|                               | YEAR        |             |
|                               | 1983        | 1982        |
| -----                         |             |             |
| UTILITY PLANT                 | \$1,430,903 | \$1,294,611 |
| LESS ACCUMULATED DEPRECIATION | \$249,504   | \$213,225   |
|                               | -----       | -----       |
| TOTAL PLANT-NET               | \$1,181,399 | \$1,081,386 |
|                               | -----       | -----       |
| INVESTMENTS                   | \$818       | \$5,639     |
| CURRENT ASSETS                |             |             |
| CASH                          | \$4,938     | \$4,200     |
| ACCOUNTS RECEIVABLE-NET       | \$28,052    | \$23,758    |
| INTEREST RECEIVABLE           | \$15,945    | \$10,206    |
| FUEL INVENTORY                | \$35,158    | \$45,643    |
| MATERIALS AND SUPPLIES        | \$16,099    | \$12,909    |
| OTHER                         | \$1,264     | \$1,743     |
|                               | -----       | -----       |
| TOTAL CURRENT ASSETS          | \$101,456   | \$98,459    |
|                               | -----       | -----       |
| DEFERRED DEBITS               | \$30,294    | \$17,459    |
|                               | -----       | -----       |
| TOTAL ASSETS                  | \$1,313,967 | \$1,202,943 |
|                               | =====       | =====       |

\*\*\*PRELIMINARY ASSET SHEET BASED ON UNAUDITED FIGURES \*\*\*

## Creating Rows From Data

A normal TABLE request sorts the lines of the report according to the BY phrase you use. The data retrieved is either sorted low-to-high or high-to-low, as requested. The lines may be limited by a screening phrase to a specific subset, but:

- They appear in a sort order.
- Lines appear only for values that are retrieved from the file.
- You can only insert free text between the rows when a sort field changes value, such as:

`ON DIVISION SUBFOOT`

- You can only insert calculations between rows when a sort field changes value, such as:

`ON DIVISION RECAP`

In contrast, the FML FOR phrase allows you to structure your report row-by-row. This organization gives you greater control over the data that is incorporated into a report and over its presentation. You can:

- Report on specific data values for a field in a data source and combine particular data values under a common label, for use in calculations.
- Type data directly into the request to supplement data that is retrieved from the data source.
- Include text, underlines, and calculations at points in the report that are not related to sort breaks.
- Perform recursive processing in which the result of an interim calculation is saved and then used as the starting point for a subsequent calculation.
- Suppress the display of rows for which no data is retrieved.
- Identify rows by labels and columns by numbers so that you can point to the individual cells formed at each intersection (as on a spreadsheet).

Syntax

How to Specify Rows

The syntax for specifying a fixed set of rows is

```
FOR fieldname [NOPRINT]
value [OR value OR...] OVER
.
.
[value OR value]
END
```

where:

*fieldname*

Is a field name in the data source.

*value*

Is the value describing the data that is retrieved for this row of the report.

Note that a tag value for a FOR field (like 1010) may be referred to only once in an FML request. However, once the data is retrieved and stored with a unique tag, you can use the retrieved values more than once.

Example

Creating Rows From Data

Assume you have a simple data source with financial data for each corporate account, as follows:

CHART OF ACCOUNTS

| ACCOUNT | DESCRIPTION         |
|---------|---------------------|
| 1010    | CASH ON HAND        |
| 1020    | DEMAND DEPOSITS     |
| 1030    | TIME DEPOSITS       |
| 1100    | ACCOUNTS RECEIVABLE |
| 1200    | INVENTORY           |
| .       | .                   |
| .       | .                   |
| .       | .                   |

Using the FOR phrase in FML, you can issue the following TABLE request in which each value of ACCOUNT is represented by a tag (1010, 1020, etc.) and displays as a separate row:

```
TABLE FILE LEDGER
SUM AMOUNT
FOR ACCOUNT
1010 OVER
1020 OVER
1030 OVER
1100 OVER
1200
END
```

The output is:

|      | AMOUNT |
|------|--------|
|      | -----  |
| 1010 | 8,784  |
| 1020 | 4,494  |
| 1030 | 7,961  |
| 1100 | 18,829 |
| 1200 | 27,307 |

## Changing Row Titles

Tags identify the data values of the FOR field in an FML report. Using the AS phrase, you can assign a row title that is different from the tag to each row of the report.

### Example

### Changing Row Titles

In the following example, the row titles CASH ON HAND and DEMAND DEPOSITS provide meaningful identifications for the corresponding tags.

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND' OVER
1020 AS 'DEMAND DEPOSITS'
END
```

Note that single quotation marks are necessary only if the row title you are assigning has embedded blanks.

The output is:

|                 | AMOUNT |
|-----------------|--------|
|                 | -----  |
| CASH ON HAND    | 8,784  |
| DEMAND DEPOSITS | 4,494  |

If no AS phrase is included, the first value will be displayed as a label in the report.

# Creating Rows From Multiple Records

There are different ways to combine multiple records from your data sources into an FML report row. You can use:

- The OR phrase to sum the values of two or more tags in a single expression.
- The TO phrase to identify a range of values on which to report.
- A mask to specify a group of tag values without having to name each one.

A value retrieved from multiple records can only be included in a single row.

Multiple tags referenced in any of these ways are evaluated first for an exact reference or for the end points of a range, next for a mask, and finally within a range. For example, if a value is specified as an exact reference and then as part of a range, the exact reference is displayed. Note that the result will be unpredictable if a value fits into more than one row whose tags have the same priority (for example, an exact reference and the end point of a range.)

In addition to these methods, you can extract multiple tags for a row from an external file.

## Syntax

### How to Sum Values in Rows With the OR Phrase

To sum the values of two or more tags in a single report row, use the OR phrase in the FOR phrase. The syntax is:

```
FOR fieldname
tag1 OR tag2 [OR tagn...]
.
.
.
```

where:

```
fieldname
```

Is a field name in the data source.

```
tag1, tag2, tagn
```

Are the tags whose values are to be summed.

## Example

### Summing Values in Rows

The following model sums the values of three tags (1010, 1020, 1030) as CASH.

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 OR 1020 OR 1030 AS 'CASH' OVER
1100 AS 'ACCOUNTS RECEIVABLE' OVER
1200 AS 'INVENTORY'
END
```

The output is:

|                     | AMOUNT |
|---------------------|--------|
|                     | -----  |
| CASH                | 21,239 |
| ACCOUNTS RECEIVABLE | 18,829 |
| INVENTORY           | 27,307 |

**Syntax****How to Identify a Range of Values With the TO Phrase**

Instead of using a specific tag for a report line, you can identify a range of tag values by including the TO phrase within the FOR phrase. The syntax is

```
tagvalue1 TO tagvalue2
```

where:

```
tagvalue1
```

Is the lower limit of the range.

```
TO
```

Is the required phrase.

```
tagvalue2
```

Is the upper limit of the range.

**Example****Identifying a Range of Values**

Since CASH accounts in the LEDGER system are accounts 1010, 1020, 1030, you can specify the range 1010 to 1030:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 TO 1030 AS 'CASH'
END
```

**Syntax****How to Use Masking Tags**

If the tag field has a character (alphanumeric) format, you can perform a masked match. Use the dollar sign character (\$) as the mask. For instance, the tag

```
A$$D
```

matches any four-character value beginning with A and ending with D. The two middle places can be any character. This is useful for specifying a whole group of tag values without having to name each one.

**Example****Using Masking Tags to Match a Group of Tags**

In this example the amounts associated with all four-character accounts that begin with 10, expressed with a mask as 10\$\$, will be used to produce the CASH row of the report.

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
10$$ AS 'CASH' OVER
1100 AS 'ACCOUNTS RECEIVABLE' OVER
1200 AS 'INVENTORY'
END
```



## Example

### Using Tags From External Files

In this example, the tags for a row of the FML report come from an external file called CASHSTUF, which contains the tags:

```
1010
1020
1030
```

The following TABLE request uses the tags from the external file, summing the amounts in accounts 1010, 1020, and 1030 into the CASH row of the FML report:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
(CASHSTUF) AS 'CASH' OVER
1100 AS 'ACCOUNTS RECEIVABLE'
END
```

Notice that the file name must be enclosed in parentheses.

## Using the BY Phrase in FML Requests

Only one FOR phrase is permitted in a TABLE request. It substitutes in part for a BY phrase, which controls the sort sequence. However, the request can also include up to 32 BY phrases. In general, BY phrases specify the major (outer) sort fields in FML reports, and the FOR phrase specifies the minor (inner) sort field.

Example

Combining BY and FOR Phrases in an FML Request

In this example the report results for ACCOUNT (the inner sort field) are sorted by REGION (the outer sort field):

```
DEFINE FILE REGION
CUR_YR=E_ACTUAL;
LAST_YR=.831*CUR_YR;
REGION/A4=IF E_ACTUAL NE 0 OR E_BUDGET NE 0 THEN 'EAST' ELSE 'WEST';
END

TABLE FILE REGION
HEADING CENTER
"CURRENT ASSETS FOR REGION <REGION>"
" "
SUM CUR_YR LAST_YR
BY REGION NOPRINT AND PAGE-BREAK
FOR ACCOUNT
10$$ AS 'CASH' OVER
1100 AS 'ACCOUNTS RECEIVABLE' OVER
1200 AS 'INVENTORY' OVER
BAR OVER
RECAP CUR_ASSET/I5C = R1 + R2 + R3;
END
```

The output is:

| CURRENT ASSETS FOR REGION EAST |          |          |
|--------------------------------|----------|----------|
|                                | CUR_YR   | LAST_YR  |
|                                | -----    | -----    |
| CASH                           | 9,511.00 | 7,903.64 |
| ACCOUNTS RECEIVABLE            | .        | .        |
| INVENTORY                      | .        | .        |
|                                | -----    | -----    |
| CUR_ASSET                      | 9,511    | 7,903    |

Note that a new report page is produced for each region in the company.

A sort field value can be used in a RECAP command to allow the model to take different actions within each major sort break. For instance, the following calculation would compute a non-zero value only for the EAST region:

```
RECAP X=IF REGION EQ 'EAST' THEN .25*CASH ELSE 0; AS 'AVAILABLE FOR DIVIDENDS'
```

For more information, see *Performing Inter-Row Calculations* on page 16-12.

# Performing Inter-Row Calculations

The RECAP command allows you to perform calculations on data in the rows of the report to produce new rows. You must supply the name and format of the field that will receive the result of the calculation, and an expression that defines the calculation you wish to perform. Since RECAP calculations are performed among rows, each row in the calculation must be uniquely identified. FML supplies default row labels for this purpose (R1, R2, etc). However, you may assign more meaningful labels. For details, see *Referring to Rows* on page 16-13.

## Syntax

### How to Define Inter-Row Calculations

```
RECAP fieldname[/format]=expression;
[AS 'text']
```

where:

**RECAP**

Is the command name and is required. It should begin on a line by itself.

*fieldname*

Is the name you assign to the calculated value. The name can be up to 66 characters long, and must start with an alphabetic character. This field name can also serve as an explicit label. See *Referring to Rows* on page 16-13.

*format*

Is the field's USAGE format. It cannot exceed the column width. The default is the format of the column in which the calculated value will be displayed.

*expression*

Can be any calculation available with the DEFINE command (including IF... THEN ... ELSE syntax, special functions, and user-written subroutines; excluding DECODE, EDIT, and fields in date format). The expression may extend to as many lines as it requires; a semicolon is required at the end of the expression. For related information, see *Using Subroutines in Calculations* on page 16-22 and the *Using Functions* manual.

The expression can include references to specific rows using the default FML positional labels (R1, R2, etc) or it can refer to rows, columns, and cells using a variety of flexible notation techniques. For details, see *Referring to Rows* on page 16-13, *Referring to Columns* on page 16-16, and *Referring to Cells* on page 16-20.

AS '*text*'

Allows you to assign a different name to the RECAP expression for the report. Enclose the text in single quotation marks.

**Reference****Usage Notes for RECAP**

- RECAP expressions refer to other rows in the model using their labels (either explicit or default). Labels referred to in a RECAP expression must also be specified in the report request.
- The format specified for the RECAP result overrides the format of the column. In the following example,

```
RECAP TOTVAL/D6.2S=IF R1 GT R4 THEN R4 ELSE R1;
AS 'REDUCED VALUE'
```

TOTVAL/D6.2S displays the result as six positions with two decimal places (and displays blanks if the value was zero) in each column of the report, regardless of the format of the data in the column. One use of this feature might be to display percentages in a column of whole numbers.

- Subtotals are not supported in FML.
- In environments that support the RETYPE command, note that RETYPE does not recognize labels in FML with field format redefinition.

## Referring to Rows

FML assigns a default positional label to each tagged, DATA, RECAP, and PICKUP row. These positional labels are automatically prefixed with the letter R, so that the first such row in the model is R1, the second is R2, etc. You can use these labels to refer to rows in RECAP expressions. (Default labels are not assigned to rows that contain underlines, blank lines, or free text since these row types do not need to be referenced in expressions.)

When you refer to rows in a RECAP expression, you can:

- Use the positional row label assigned by FML.
- Create an explicit row label of your own.
- Mix positional and explicit row labels.

If you assign an explicit label, the positional label (R1, R2, etc.) is retained internally.

Note that an explicit label is not needed for a RECAP row because the field name on the left of the equal sign can be used as a label.

Syntax

How to Assign an Explicit Row Label

`rowtype AS 'text' LABEL label OVER`

where:

`rowtype`

Can be a tag, DATA, or PICKUP row.

`AS 'text'`

Allows you to assign a different name to the row for the report. Enclose the text in single quotation marks.

`label`

Can be up to 66 characters and cannot have blanks or special characters. Each explicit label you assign must be unique.

Example

Referring to Default Row Labels in RECAP Expressions

In this example, FML assigns account 1010 the implicit label R1; account 1020, the implicit label R2; and account 1030, the implicit label R3. Since no label is assigned to a BAR row, the RECAP row is assigned the implicit label R4.

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND' OVER
1020 AS 'DEMAND DEPOSITS' OVER
1030 AS 'TIME DEPOSITS' OVER
BAR OVER
RECAP TOTCASH = R1 + R2 + R3; AS 'TOTAL CASH'
END
```

The output is:

|                 | AMOUNT |
|-----------------|--------|
|                 | -----  |
| CASH ON HAND    | 8,784  |
| DEMAND DEPOSITS | 4,494  |
| TIME DEPOSITS   | 7,961  |
|                 | -----  |
| TOTAL CASH      | 21,239 |

Example Referring to Explicit Row Labels in RECAP Expressions

The following request assigns the labels CASH, AR, and INV to three tag rows, which are referenced in the RECAP expression.

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
10$$$ AS 'CASH' LABEL CASH OVER
1100 AS 'ACCOUNTS RECEIVABLE' LABEL AR OVER
1200 AS 'INVENTORY' LABEL INV OVER
BAR OVER
RECAP CURASST/I5C= CASH + AR + INV;
END
```

The output is:

|                     |        |
|---------------------|--------|
|                     | AMOUNT |
|                     | -----  |
| CASH                | 21,239 |
| ACCOUNTS RECEIVABLE | 18,829 |
| INVENTORY           | 27,307 |
|                     | -----  |
| CURASST             | 67,375 |

Note that the RECAP command could subsequently be referred to by the name CURASST, which functions as an explicit label:

```
RECAP CURASST/I5=CASH+AR+INV;
```

Example Using Labels to Repeat Rows

In certain cases, you may wish to repeat an entire row later in your report. For example, the CASH account can appear in the Asset statement and Cash Flow statements of a financial analysis, as shown below:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
"ASSETS" OVER
10$$$ AS 'CASH' LABEL TOTCASH OVER
.
.
"CASH FLOW" OVER
RECAP SAMECASH/I5C=TOTCASH; AS 'CASH'
END
```

When you refer to the CASH row the second time, you can use a RECAP calculation (with a new name) and refer to the label, either explicitly (TOTCASH) or implicitly (R1), in the row where CASH was first used.

# Referring to Columns

- An FML report can refer to explicit columns as well as explicit rows. You can refer to columns using:
- Column numbers.
  - Contiguous column notation in RECAP expressions--for example (2,5) to represent columns 2 through 5.
  - Column addressing.
  - A factor to represent every other column, or every third column, etc.
  - Column values.

## Referring to Column Numbers

A calculation may be performed for one column or for a specific set of columns. To identify the columns, you place the column number in parentheses after the label name.

### Example

### Referring to Column Numbers

```
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
END

TABLE FILE LEDGER
SUM CUR_YR AS 'CURRENT,YEAR'
LAST_YR AS 'LAST,YEAR'
FOR ACCOUNT
1010 AS 'CASH ON HAND' OVER
1020 AS 'DEMAND DEPOSITS' OVER
1030 AS 'TIME DEPOSITS' OVER
BAR OVER
RECAP TOTCASH/I5C = R1 + R2 + R3; AS 'TOTAL CASH' OVER
" " OVER
RECAP GROCASH(2)/F5.2=100*TOTCASH(1)/TOTCASH(2) - 100;
AS 'CASH GROWTH(%)'
END
```

- In the second RECAP expression, note that:
- TOTCASH(1) refers to total cash in column 1.
  - TOTCASH(2) refers to total cash in column 2.
  - The resulting calculation is displayed in column 2 of the row labeled CASH GROWTH(%).
- The RECAP is only calculated for the column specified.

The output is:

|                 | CURRENT<br>YEAR | LAST<br>YEAR |
|-----------------|-----------------|--------------|
|                 | -----           | -----        |
| CASH ON HAND    | 8,784           | 7,214        |
| DEMAND DEPOSITS | 4,494           | 3,482        |
| TIME DEPOSITS   | 7,961           | 6,499        |
|                 | -----           | -----        |
| TOTAL CASH      | 21,239          | 17,195       |
| CASH GROWTH(%)  |                 | 23.52        |

After data retrieval is completed, a single column is calculated all at once, and multiple columns one by one.

Referring to Contiguous Columns

When a set of contiguous columns is needed within a RECAP, you can separate the first and last column numbers with commas. For example, DIFFERENCE (2,5) indicates that you want to compute the results for columns 2 through 5.

Example

Recapping Over Contiguous Columns

In this example the RECAP calculation for ATOT occurs only for columns 2 and 3, as specified in the request. No calculation is performed for column 1.

```
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
NEXT_YR/I5C=1.13*CUR_YR + 222;
END

TABLE FILE LEDGER
SUM NEXT_YR CUR_YR LAST_YR
FOR ACCOUNT
10$$ AS 'CASH' OVER
1100 AS 'ACCOUNTS RECEIVABLE' OVER
1200 AS 'INVENTORY' OVER
BAR OVER
RECAP ATOT(2,3)/I5C = R1 + R2 + R3;
AS 'ASSETS--ACTUAL'
END
```

The output is:

|                     | NEXT_YR | CUR_YR | LAST_YR |
|---------------------|---------|--------|---------|
|                     | -----   | -----  | -----   |
| CASH                | 25,991  | 21,239 | 17,195  |
| ACCOUNTS RECEIVABLE | 21,941  | 18,829 | 15,954  |
| INVENTORY           | 31,522  | 27,307 | 23,329  |
|                     | -----   | -----  | -----   |
| ASSETS--ACTUAL      |         | 67,375 | 56,478  |



## Referring to Column Addresses

When you need a calculation not for every column, but for every other, or every third column, you can supply a factor, or column address, to do this. Column addressing is particularly useful when several data fields are displayed within each value of a column sort.

### Syntax

#### How to Use Column Addressing

The left-hand side of the expression has the form

`fieldname(s,e,i)[/format]=`

where:

`fieldname`

Is the name you assign to the calculated value.

`s`

Is the starting column.

`e`

Is the ending column (may be \* to denote all columns).

`i`

Is the increment factor.

`format`

Is the field's USAGE format. The default value is the format of the original column.

### Example

#### Applying Column Addressing

In the following statement, there are two columns for each month:

`SUM ACTUAL AND FORECAST ACROSS MONTH`

If you want to perform a calculation only for the ACTUAL data, you can control the placement of the results with a RECAP in the form:

`RECAP VALUE(1,*,2)=expression;`

The asterisk means to continue the RECAP for *all* odd-numbered columns (beginning in column 1, with an increment of 2, for all columns).

## Referring to Relative Column Addresses

A calculation can refer to a column plus or minus 1, 2, etc., relative to a starting point. You can use an asterisk (\*) to indicate “this column”—that is, the column to which the reference is being made. For example,

```
COMP=FIX(*)-FIX(*-1);
```

can refer to the change in fixed assets from one period to the next. The reference to COMP=FIX(\*) is equivalent to COMP=FIX;. When referring to a prior column, it must have already been retrieved or its value is zero.

### Example

### Applying Relative Column Addressing

This example computes the change in cash (CHGCASH) for columns 1 and 2.

```
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
NEXT_YR/I5C=1.13*CUR_YR + 222;
END

TABLE FILE LEDGER
SUM NEXT_YR CUR_YR LAST_YR
FOR ACCOUNT
10$$ AS 'TOTAL CASH' LABEL TOTCASH OVER
" " OVER
RECAP CHGCASH (1,2)/I5C = TOTCASH(*) - TOTCASH(*+1);
AS 'CHANGE IN CASH' OVER
END
```

The output is:

|                | NEXT_YR | CUR_YR | LAST_YR |
|----------------|---------|--------|---------|
|                | -----   | -----  | -----   |
| TOTAL CASH     | 25,991  | 21,239 | 17,195  |
| CHANGE IN CASH |         | 4,752  | 4,044   |

## Referring to Column Values

When a report is sorted using the ACROSS phrase, all of the retrieved values are aligned under their appropriate columns. Each column has a title consisting of one value of the ACROSS field. The entire column of data can be directly addressed by this value in a RECAP calculation.

### Example

#### Referring to a Column by Its Value

The following request would produce a report with YEAR as the heading across the top of the report, and year values (19nn to 19nn) as column titles below it:

```
SUM AMOUNT ACROSS YEAR
```

```
.
. .
.
```

You could then issue a RECAP command like the following for specific columns of data:

```
RECAP FACTOR=IF YEAR LT 1975 THEN .09*COST ELSE ESTCOST;
```

## Referring to Cells

You can refer to columns and rows using a form of cell notation that identifies the intersection of a row and a column as (r, c).

### Syntax

#### How to Use Cell Notation for Rows and Columns

A row and column can be addressed in an expression by the notation

```
E(r,c)
```

where:

*E*

Is a required constant.

*r*

Is the row number.

*c*

Is the column number. Use an asterisk (\*) to indicate the current column.

Example

Referring to Columns Using Cell Notation

In this request, two RECAP expressions derive VARIANCEs (EVAR and WVAR) by subtracting values in four columns (1, 2, 3, 4) in row three (PROFIT); these values are identified using cell notation (r,c).

```
TABLE FILE REGION
SUM E_ACTUAL E_BUDGET W_ACTUAL W_BUDGET
FOR ACCOUNT
3000 AS 'SALES' OVER
3100 AS 'COST' OVER
BAR OVER
RECAP PROFIT/I5C = R1 - R2; OVER
" " OVER
RECAP EVAR(1)/I5C = E(3,1) - E(3,2);
AS 'EAST--VARIANCE' OVER
RECAP WVAR(3)/I5C = E(3,3) - E(3,4);
AS 'WEST--VARIANCE'
END
```

The output is:

|                | E_ACTUAL | E_BUDGET | W_ACTUAL | W_BUDGET |
|----------------|----------|----------|----------|----------|
|                | -----    | -----    | -----    | -----    |
| SALES          | 6,000    | 4,934    | 7,222    | 7,056    |
| COST           | 4,650    | 3,760    | 5,697    | 5,410    |
|                | -----    | -----    | -----    | -----    |
| PROFIT         | 1,350    | 1,174    | 1,525    | 1,646    |
| EAST--VARIANCE | 176      |          |          |          |
| WEST--VARIANCE |          |          | -121     |          |

**Note:** In addition to illustrating cell notation, this example demonstrates the use of column numbering. Notice that the display of the EAST and WEST VARIANCEs in columns 1 and 3, respectively, are controlled by the numbers in parentheses in the request: EVAR (1) and WVAR (3).

## Using Subroutines in Calculations

You may provide your own calculation routines in RECAP rows to perform special-purpose calculations, a useful feature when these calculations are mathematically complex or require extensive look-up tables.

User-written routines are coded as subroutines in any language that supports a call process, such as FORTRAN, COBOL, PL/1, and BAL. See the *Using Functions* manual for information about creating your own subroutines.

### Syntax

#### How to Call User-Written Subroutines in RECAP Commands

```
RECAP calcname[(s,e,i)][/format] = subroutine (input1,...,inputn, 'format2');
```

where:

*calcname*

Specifies the name of the calculation. You also may specify a start (s), end (e), and increment (i) value for the column where you want the value displayed; if omitted the value appears in all columns.

*format*

The format for the calculation is optional; the default is the format of the column. If the calculation consists of only the subroutine, make sure that the format of the subroutine output value (*format2*) agrees with the calculation's format. If the calculation format is larger than the column width, the value displays in that column as asterisks.

*subroutine*

Is the eight-character name of the subroutine. It must be different from any row label and cannot contain any of the following special characters: =, / ( ).

*input*

Are the input arguments for the call to the subroutine; they may include numeric constants, alphanumeric literals, row and column references (R notation, E notation, or labels), or names of other RECAP calculations.

Make sure that the values being passed to the subroutine agree in number and type with the arguments as coded in the subroutine.

*format2*

Is the format of the return value, which must be enclosed in single quotation marks.

**Example****Calling a Subroutine in a RECAP Command**

Suppose you have a subroutine named INVEST in your private collection of subroutines (INVEST is not available in the supplied library) and it calculates the amount on the basis of cash on hand, total assets, and the current date. In order to create a report that prints an account of company assets and calculates how much money the company has available to invest, you must create a report request that invokes the INVEST subroutine.

The current date is obtained from the &YMD system variable. The NOPRINT option beside it prevents the date from appearing in the report; the date is solely used as input for the next RECAP statement.

The request is:

```
TABLE FILE LEDGER
HEADING CENTER
"ASSETS AND MONEY AVAILABLE FOR INVESTMENT </2"
SUM AMOUNT ACROSS HIGHEST YEAR
IF YEAR EQ 1985 OR 1986
FOR ACCOUNT
1010 AS 'CASH' LABEL CASH OVER
1020 AS 'ACCOUNTS RECEIVABLE' LABEL ACR OVER
1030 AS 'INTEREST RECEIVABLE' LABEL ACI OVER
1100 AS 'FUEL INVENTORY' LABEL FUEL OVER
1200 AS 'MATERIALS AND SUPPLIES' LABEL MAT OVER
BAR OVER
RECAP TOTCAS = CASH+ACR+ACI+FUEL+MAT; AS 'TOTAL ASSETS' OVER
BAR OVER
RECAP THISDATE/A8 = &YMD; NOPRINT OVER
RECAP INVAI = INVEST(CASH,TOTCAS,THISDATE,'D12.2'); AS
 'AVAIL. FOR INVESTMENT' OVER
BAR AS '='
END
```

The output is:

PAGE 1

ASSETS AND MONEY AVAILABLE FOR INVESTMENT

|                        | YEAR   |        |
|------------------------|--------|--------|
|                        | 1986   | 1985   |
| CASH                   | 2,100  | 1,684  |
| ACCOUNTS RECEIVABLE    | 875    | 619    |
| INTEREST RECEIVABLE    | 4,026  | 3,360  |
| FUEL INVENTORY         | 6,250  | 5,295  |
| MATERIALS AND SUPPLIES | 9,076  | 7,754  |
|                        | -----  | -----  |
| TOTAL ASSETS           | 22,327 | 18,712 |
|                        | -----  | -----  |
| AVAIL. FOR INVESTMENT  | 3,481  | 2,994  |
|                        | =====  | =====  |

# Supplying Data Directly in the FML Request

In certain cases, you may need to include some additional constants (for example, exchange rates, inflation rates, etc.) in your model. Not all data values for the model have to be retrieved from the data source. Using FML, you can supply data directly in the request.

## Syntax

### How to Supply Data Directly in a Request

```
DATA value,[..., value],$
[AS 'text'] [LABEL label]
```

where:

*value*

Specifies the values that you are supplying. Values in a list must be separated by commas. The list must end with a comma and a dollar sign (\$).

*AS 'text'*

Allows you to assign a different title to the data value. Enclose the text in single quotation marks.

*label*

Assigns a name to the data for use in RECAP calculations. The label can be up to 66 characters and cannot have blanks or special characters. Each explicit label you assign must be unique.

## Example

### Supplying Data Directly in a Request

In this example, two values (.53 and 1.37) are provided for the exchange rates of marks and pounds, respectively:

```
DEFINE FILE LEDGER
MARKS/I5C=AMOUNT;
POUNDS/I5C=3.2*AMOUNT;
END

TABLE FILE LEDGER
SUM MARKS AS 'GERMAN,DIVISION'
POUNDS AS 'ENGLISH,DIVISION'
FOR ACCOUNT
1010 AS 'CASH--LOCAL CURRENCY' LABEL CASH OVER
DATA .53 , 1.37 ,$ AS 'EXCHANGE RATE' LABEL EXCH OVER
RECAP US_DOLLARS/I5C= CASH * EXCH;
END
```

The values supplied are taken one column at a time for as many columns as the report originally specified.

The output is:

|                      | GERMAN<br>DIVISION | ENGLISH<br>DIVISION |
|----------------------|--------------------|---------------------|
|                      | -----              | -----               |
| CASH--LOCAL CURRENCY | 8,784              | 28,106              |
| EXCHANGE RATE        | .53                | 1.37                |
| US_DOLLARS           | 4,655              | 38,505              |

# Inserting Rows of Free Text

You can insert text anywhere in your FML report by typing it on a line by itself and enclosing it within double quotation marks. You can also add blank lines, designated as text, to improve the appearance of the report.

In addition, you can include data developed in your FML report in a row of free text by including the label for the data variable in the text row.

## Example

### Inserting Free Text

In this example, three rows of free text are inserted, one blank and two text rows:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
" --- CASH ACCOUNTS ---" OVER
1010 AS 'CASH ON HAND' OVER
1020 AS 'DEMAND DEPOSITS' OVER
1030 AS 'TIME DEPOSITS' OVER
" " OVER
" --- OTHER CURRENT ASSETS ---" OVER
1100 AS 'ACCOUNTS RECEIVABLE' OVER
1200 AS 'INVENTORY' OVER
END
```

The output is:

|                              | AMOUNT |
|------------------------------|--------|
|                              | -----  |
| --- CASH ACCOUNTS ---        |        |
| CASH ON HAND                 | 8,784  |
| DEMAND DEPOSITS              | 4,494  |
| TIME DEPOSITS                | 7,961  |
| ---                          |        |
| --- OTHER CURRENT ASSETS --- |        |
| ACCOUNTS RECEIVABLE          | 18,829 |
| INVENTORY                    | 27,307 |

Notice that the blank row was created by enclosing a blank within double quotation marks on a separate line of the report request.



## Syntax

### How to Insert Data Variables In Text Rows

`"text <label[(c)][>]"`

where:

`<`

Is a required left caret to bracket the label.

`label`

Is the explicit or implicit row label.

`c`

Is an optional cell identifier that indicates the column number of the cell. This identifier, however, is required whenever there is more than one column in the report. If you use it, enclose it in parentheses.

`>`

Is an optional right bracket that can be used to make the positioning clearer.

For related information, see Chapter 9, *Customizing Tabular Reports*.

## Example

### Inserting a Data Variable in a Text Row

The following text line retrieves the value contained in column 1 for the row labeled TOTCASH:

`"TOTAL CASH IN THE CURRENT YEAR IS <TOTCASH(1)"`

## Adding Columns to an FML Report

The request controls the number of columns in any report. For instance, if a request contains the display command SUM AMOUNT AND FORECAST, the report will contain two columns: AMOUNT and FORECAST.

You can add columns in an FML request, just as you can in a TABLE request, using the COMPUTE command to calculate a value or simply to allocate the space, column title, and format for a column. For related information, see Chapter 6, *Creating Temporary Fields*.

### Example

#### Adding Columns to an FML Report

This example performs the designated calculation on each tagged or RECAP row of the report. The RECAP rows, however, may change the calculation.

```
SUM CUR_YR AND LAST_YR
COMPUTE DIFFERENCE/D8S=CUR_YR - LAST_YR ;
FOR ACCOUNT
.
.
.
END
```

To add one or more future time periods to a report:

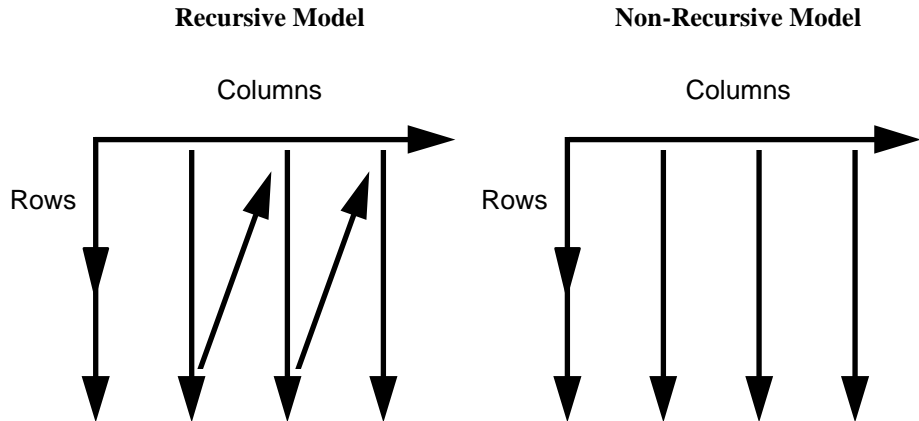
```
SUM AMOUNT ACROSS YEAR AND COMPUTE 1999/D12.2S=;
WHERE YEAR GE '1992' AND YEAR LE '1998'
.
.
.
```

## Creating Recursive Models

Models involving different time periods often require using the ending value of one time period as the starting value for the next time period. The series of calculations describing these situations has two characteristics:

- The labels on one or more RECAP rows are duplicates of other rows. That is, they are used repeatedly to recompute some values.
- A calculation may refer to a label not yet described, but provided later in the model. If, at the end of the model, a label that is needed is missing, an error message is displayed.

Recursive models require that the columns are produced in sequential order, one by one. In nonrecursive models, all of the columns can be produced simultaneously. Schematically, these patterns are shown below.



FML automatically switches to sequential order as soon as either of the two modeling conditions requiring the switch is recognized (that is, either reuse of labels by different rows, or forward reference to a label in a calculation).

### Example

### Recursive Models

The following example illustrates recursive models. Note that one year's ENDCASH becomes the next year's STARTING CASH.

```

DEFINE FILE REGION
CUR_YR=E_ACTUAL;
LAST_YR=.831*CUR_YR;
NEXT_YR=1.2297*CUR_YR;
END

TABLE FILE REGION
SUM LAST_YR CUR_YR NEXT_YR
FOR ACCOUNT
10$$ AS 'STARTING CASH' LABEL STCASH OVER
RECAP STCASH(2,*) = ENDCASH(*-1); OVER
" " OVER
3000 AS 'SALES' LABEL SLS OVER
3100 AS 'COST' LABEL COST OVER
BAR OVER
RECAP PROFIT/I5C = SLS - COST; OVER
" " OVER
RECAP ENDCASH/I5C = STCASH + PROFIT;
END

```

The output is:

PAGE 1

|               | LAST_YR | CUR_YR | NEXT_YR |
|---------------|---------|--------|---------|
|               | -----   | -----  | -----   |
| STARTING CASH | 7,903   | 9,024  | 10,374  |
| SALES         | 4,985   | 6,000  | 7,378   |
| COST          | 3,864   | 4,650  | 5,718   |
|               | -----   | -----  | -----   |
| PROFIT        | 1,121   | 1,350  | 1,660   |
| ENDCASH       | 9,024   | 10,374 | 12,034  |

**Note:** Under CMS, the above example generates a warning message.

# Formatting an FML Report

You can improve the readability and presentation of your FML report by adding underlines and page breaks.

- **Underlining.** Reports with columns of numbers frequently need to display underlines before some RECAP calculations. You can specify an underline character introduced by the word BAR, in place of the tag value.
- **Page breaks.** You can request a new page at any point in a report by placing the word PAGE-BREAK in place of the tag value.

Beyond the formatting available in FML, you can style elements of your reports (free text, labels, data, RECAP calculations, and underlines) using StyleSheet syntax. For details, see the documentation on formatting reports with StyleSheets.

## Syntax

### How to Add an Underline Character for Columns

The syntax is

```
BAR [AS 'character'] OVER
```

where:

```
character
```

Is either the hyphen character (-) or the equal character (=). Enclose the character in single quotation marks. The default character is the hyphen (-).

## Example

### Underling Columns

This example uses the default underscore character (-):

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND' OVER
1020 AS 'DEMAND DEPOSITS' OVER
1030 AS 'TIME DEPOSITS' OVER
BAR OVER
RECAP TOTCASH = R1 + R2 + R3;
END
```

The output is:

|                 | AMOUNT |
|-----------------|--------|
|                 | -----  |
| CASH ON HAND    | 8,784  |
| DEMAND DEPOSITS | 4,494  |
| TIME DEPOSITS   | 7,961  |
|                 | -----  |
| TOTCASH         | 21,239 |

Notice that the BAR ... OVER phrases underline only the column containing the display field.

## Syntax

### How to Specify a Page Break in FML Reports

To specify a page break include the following syntax in the FML request in place of a tag value:

```
PAGE-BREAK OVER
```

## Suppressing Tagged Rows

You may sometimes wish to retrieve a tagged row solely for use in a calculation, without displaying the row in a report. To suppress the display of a tagged row, you can add the word NOPRINT to the row declaration, as you would in a TABLE request.

You can also suppress the display of intermediate RECAP rows used as input to other calculations by adding the word NOPRINT to the RECAP command after the semicolon.

You may also wish to suppress the display of a tagged row if no data is found for the values. For details, see *Suppressing Rows With No Data* on page 16-32.

### Example

#### Suppressing the Display of Rows

This example uses the value of COST in its computation, but does not display COST as a row in the report.

```
DEFINE FILE REGION
AMOUNT/I5C=E_ACTUAL;
END

TABLE FILE REGION
SUM AMOUNT FOR ACCOUNT
3000 AS 'SALES' LABEL SLS OVER
3100 AS 'COST' LABEL COST NOPRINT OVER
RECAP PROFIT/I5C = SLS - COST; OVER
" " OVER
RECAP ROS/F6.2=100*PROFIT/SLS;
AS 'RETURN ON SALES'
END
```

The output is:

|                 | AMOUNT |
|-----------------|--------|
|                 | -----  |
| SALES           | 6,000  |
| PROFIT          | 1,350  |
| RETURN ON SALES | 22.50  |

## Suppressing Rows With No Data

The text for a tagged row is displayed even if no data is found in the file for the tag values, with a period (.) representing the missing data. You can override this convention by adding the phrase WHEN EXISTS to the definition of a tagged row. This makes displaying a row dependent upon the existence of data for the tag. This feature is particularly useful, for example, when the same model is applied to different divisions in a company.

### Example

#### Suppressing Rows With No Data

In this example, assume that the variable DIVISION contains Division 1, a real estate syndicate, and Division 2, a bank. The following request describes their balance sheets in one FML report. Rows that are irrelevant for each division will not be displayed.

```
TABLE FILE LEDGER
HEADING CENTER
"BALANCE SHEET FOR DIVISION <DIVISION>"
" "
SUM AMOUNT
BY DIVISION NOPRINT AND PAGE-BREAK
FOR ACCOUNT
2000 AS 'LAND' WHEN EXISTS LABEL LD OVER
2100 AS 'CAR LOANS' WHEN EXISTS LABEL LOAN OVER
.
.
.
```

## Saving and Retrieving Intermediate Report Results

Many reports require results developed in prior reports. This can be accomplished only if a place is provided for storing intermediate values. An example is the need to compute net profit in an Income Statement prior to calculating equity in a Balance Sheet. FML can save selected rows from one or more models by posting them to a work file. The posted rows can then be picked up from the work file and reused.

The default work file is FOCPOST. This is a comma-delimited file from which you can report directly if a FOCPOST Master File is available. In order to use the work file in a request, you must assign a physical name to the FOCPOST ddname before running the report that posts the data, and again before running the report that picks up the data.

You can assign the physical name to the file by issuing a FILEDEF command on NT, UNIX, and CMS, or a TSO ALLOCATE or DYNAM ALLOCATE command on MVS, before the request is run.

While you cannot prepare an FML report entirely from data that you supply directly in your request, you can, if you wish, prepare a report entirely from data that is stored in a comma-delimited work file.

## Posting Data

You can save any tagged, RECAP, or DATA row by posting the output to a file. These rows can then be used as though they were provided in a DATA row.

The row will be processed in the usual manner in the report, depending on its other options, and then posted. The label of the row is written first, followed by the numeric values of the columns, each comma-separated, and ending with the terminator character (\$). For an illustration, see *Posting Rows to a Work File* on page 16-33.

### Syntax

#### How to Post Data to a File

The syntax for saving any tag, RECAP, or data row is:

```
POST [TO ddname]
```

where:

*ddname*

Is the logical name you assign to the work file in which you are posting data.

You add this syntax to any row you wish to post to the work file.

### Example

#### Posting Rows to a Work File

The following request create an FML report, and posts two tag rows to the work file, LEDGEOUT:

```
FILEDEF LEDGEOUT DISK [PATH]\LEDGEOUT.DAT
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
END

TABLE FILE LEDGER
SUM CUR_YR LAST_YR
FOR ACCOUNT
1100 LABEL AR POST TO LEDGEOUT OVER
1200 LABEL INV POST TO LEDGEOUT
END
```

The output is:

|     |          |          |
|-----|----------|----------|
| AR  | , 18829, | 15954,\$ |
| INV | , 27307, | 23329,\$ |



## Syntax

### How to Pick Up Data From a Work File

You can retrieve posted rows from any work file and use them as if they were provided in a DATA row by adding the following phrase to an FML request:

```
DATA PICKUP [FROM ddname] id1 [OR id2...] [LABEL label] [AS 'text']
```

where:

*ddname*

Is the logical name of the work file from which you are retrieving data.

*id*

Is the label that was assigned in the work file to the posted row of data that is now being picked up.

*label*

Is the label you wish to assign to the data you are picking up.

The label you assign to the picked data can, but is not required to, match the id of the posted data.

You can include LABEL and AS phrases, but WHEN EXISTS is not supported.

## Example

### Picking Up Data From a Work File

In the following example, the data in the LEDGER data source and in the LEDGEOUT work file will be used in the RECAP calculation. (To see how this file was created, refer to *Posting Rows to a Work File* on page 16-33.)

#### Tip:

You must assign a logical name to the file by issuing a FILEDEF command on NT, UNIX, and CMS, or a TSO ALLOCATE or DYNAM ALLOCATE command on MVS, before the request is run.

```
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
END

TABLE FILE LEDGER
SUM CUR_YR LAST_YR
FOR ACCOUNT
1010 TO 1030 AS 'CASH' LABEL CASH OVER
DATA PICKUP FROM LEDGEOUT AR
AS 'ACCOUNTS RECEIVABLE' LABEL AR OVER
DATA PICKUP FROM LEDGEOUT INV
AS 'INVENTORY' LABEL INV OVER
BAR OVER
RECAP CUR_ASSET/I5C = CASH + AR + INV;
END
```

The output is:

|                     | CUR_YR | LAST_YR |
|---------------------|--------|---------|
|                     | -----  | -----   |
| CASH                | 21,239 | 17,195  |
| ACCOUNTS RECEIVABLE | 18,829 | 15,954  |
| INVENTORY           | 27,307 | 23,329  |
|                     | -----  | -----   |
| CUR_ASSET           | 67,375 | 56,478  |

The following line could be used to pick up the sum of the two accounts from LEDGEOUT:

```
DATA PICKUP FROM LEDGEOUT AR OR INV
AS 'ACCTS REC AND INVENTORY'
```

**Note:** Since the rows in a PICKUP file are stored in standard comma-delimited format, they could have been provided either from a prior posting, or directly by a user.

## Creating HOLD Files From FML Reports

A report created with FML can be extracted to a HOLD file in the same way as all other reports created with the TABLE language (see Chapter 11, *Saving and Reusing Report Output*). In this case, you identify the set of tag values specified for each row by the description field (the AS text supplied in the model). When no text is given for a row, the first tag value is used automatically. Therefore, in simple models with only one tag per row and no text, the lines in the HOLD file contain the single tag value. The rows derived from the RECAP calculation form part of the HOLD file. Pure text rows (including BAR rows) are omitted.

For HOLD to be supported with RECAP, the format of the RECAP field must be the same as the format of the original column.

This feature enables you to create new rows in the HOLD file that are the result of calculations. The augmented HOLD file may then be used in a variety of TABLE requests.

**Note:** RECAP rows cannot be reformatted when creating HOLD files.

Example

Creating a Hold File From an FML Report

The following request creates a HOLD file that contains records for CASH, ACCOUNTS RECEIVABLE, INVENTORY, and the RECAP row CURRENT ASSETS:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 TO 1030 AS 'CASH' OVER
1100 AS 'ACCOUNTS RECEIVABLE' OVER
1200 AS 'INVENTORY' OVER
RECAP CA = R1 + R2 + R3; AS 'CURRENT ASSETS'
ON TABLE HOLD
END
```

You can query the HOLD file:

```
>
? hold

DEFINITION OF HOLD FILE: HOLD
```

| FIELDNAME | ALIAS | FORMAT |
|-----------|-------|--------|
|           | EO1   | A19    |
| AMOUNT    | EO2   | I5C    |

You can then report from the HOLD file as:

```
TABLE FILE HOLD
PRINT E01 E02
END
```

The output is:

|                     | AMOUNT |
|---------------------|--------|
|                     | -----  |
| CASH                | 21,239 |
| ACCOUNTS RECEIVABLE | 18,829 |
| INVENTORY           | 27,307 |
| CURRENT ASSETS      | 67,375 |

---

## CHAPTER 17

# Creating a Free-Form Report

### Topics:

- Introduction to Free-Form Reports
- Designing a Free-Form Report

You can present data in an unrestricted or free-form format using a layout of your own design.

Whereas tabular and matrix reports present data in columns and rows for the purpose of comparison across records, and graphic reports present data visually using charts and graphs, free-form reports reflect your chosen positioning of data on a page.

Free-form reporting meets your needs when your goal is to present a customized picture of a data source record on each page of a report.

# Introduction to Free-Form Reports

You can design a free-form report by creating a TABLE request that omits the display commands that control columnar and matrix formatting (PRINT, LIST, SUM, and COUNT). Instead, the request includes the following report features:

|                         |                                                                                                                                                                                         |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Heading</b>          | Contains the body of the report. It displays the text characters, graphic characters, and data fields that make up the report.                                                          |
| <b>Footing</b>          | Contains the footing of the report. This is the text that appears at the bottom of each page of the report. The footing may display the same characters and data fields as the heading. |
| <b>Prefix operators</b> | Indicate field calculations and manipulation.                                                                                                                                           |
| <b>Temporary fields</b> | Derive new values from existing fields in a data source.                                                                                                                                |
| <b>BY phrases</b>       | Specify the report's sort order and determine how many records are included on each page.                                                                                               |
| <b>WHERE criteria</b>   | Select records for the report.                                                                                                                                                          |

This topic contains the following:

- Analysis of a sample free-form report and the request that generated it. See *Creating a Free-Form Report* on page 17-2.
- Description of how to design a free-form report using text, data fields, and graphic characters in the heading and footing. You will also learn how to manipulate and calculate data fields for your report. See *Designing a Free-Form Report* on page 17-6.
- Discussion of report layout using spot markers. See *Laying Out a Free-Form Report* on page 17-8.
- Discussion of sorting and record selection in a free-form report. See *Sorting and Selecting Records in a Free-Form Report* on page 17-8.

## Example

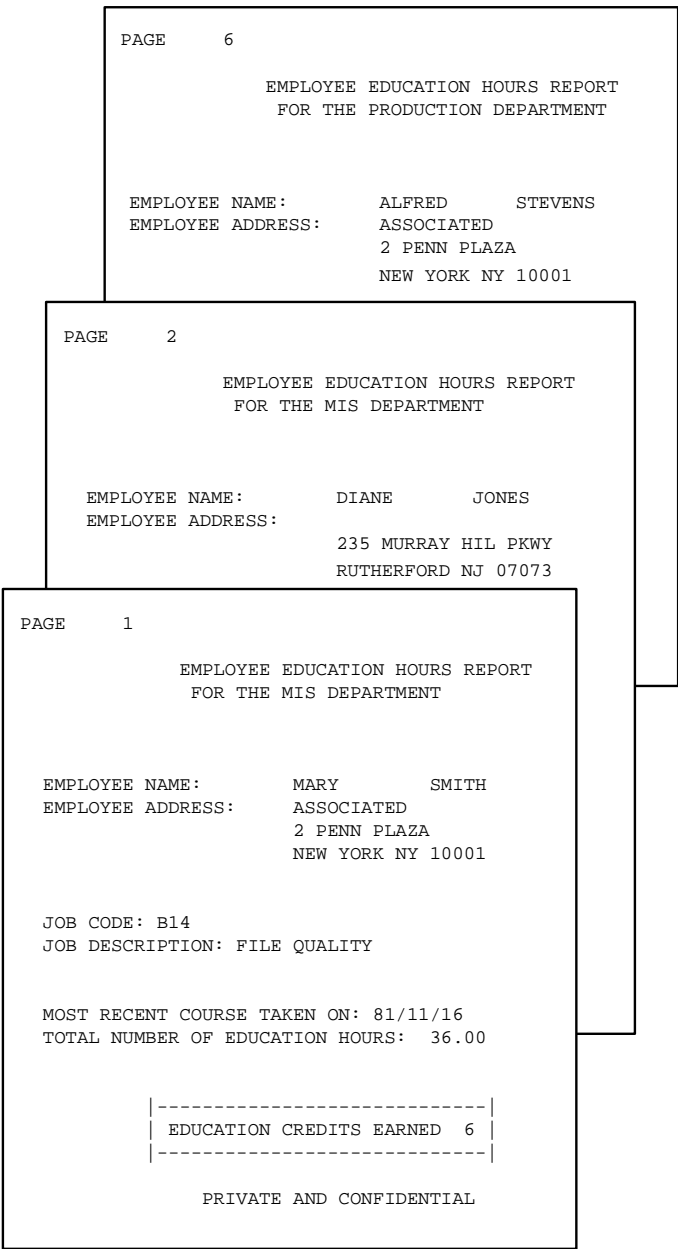
### Creating a Free-Form Report

Suppose that you are a Personnel Manager and it is your responsibility to administer your company's education policy. This education policy states that the number of hours of outside education that an employee can take at the company's expense is determined by the number of hours of in-house education completed by the employee.

To do your job efficiently, you would like a report that shows the in-house education history of each employee. Each employee's information should display on a separate page so that it can be placed in the employee's personnel file and referenced when an employee requests approval to take outside courses.

To meet this requirement, you create the EMPLOYEE EDUCATION HOURS REPORT, which displays a separate page for each employee. Notice that pages 1 and 2 of the report provide information about employees in the MIS department, while page 6 provides information for an employee in the Production department.

The following diagram simulates the output you would see if you ran the procedure in *Request for EMPLOYEE EDUCATION HOURS REPORT* on page 17-4.



## Example

### Request for EMPLOYEE EDUCATION HOURS REPORT

The following request produces the EMPLOYEE EDUCATION HOURS REPORT, which you can see in *Creating a Free-Form Report* page 17-2. Numbers to the left of the request correspond to numbers in the following annotations:

```
1. DEFINE FILE EMPLOYEE
 CR_EARNED/I2 = IF ED_HRS GE 50 THEN 9
 ELSE IF ED_HRS GE 30 THEN 6
 ELSE 3;
 END
2. TABLE FILE EMPLOYEE
3. HEADING
 "PAGE <TABPAGENO"
 " "
 "<13>EMPLOYEE EDUCATION HOURS REPORT"
4. "<14>FOR THE <DEPARTMENT DEPARTMENT"
5. "</2"
 "EMPLOYEE NAME: <FIRST_NAME> <LAST_NAME>"
 "EMPLOYEE ADDRESS: <ADDRESS_LN1>"
 "<23><ADDRESS_LN2>"
 "<23><ADDRESS_LN3>"
 "</1"
 "JOB CODE: <JOBCODE>"
 "JOB DESCRIPTION: <JOB_DESC>"
 "</1"
6. "MOST RECENT COURSE TAKEN ON: <MAX.DATE_ATTEND>"
 "TOTAL NUMBER OF EDUCATION HOURS: <ED_HRS>"
 "</1"
7. "<10>|-----|"
8. "<10>| EDUCATION CREDITS EARNED <CR_EARNED>|"
 "<10>|-----|"
9. FOOTING
 "<15>PRIVATE AND CONFIDENTIAL"
 BY DEPARTMENT
10. BY EMP_ID NOPRINT PAGE-BREAK
11. WHERE ED_HRS GT 0
 END
```

The list that follows explains the role of each line of the request in producing the sample report:

1. The DEFINE command creates a virtual field for the report. The calculation reflects the company's policy for earning outside education credits. The result is stored in CR\_EARNED and appears later in the report.
2. A free-form report begins with a standard TABLE FILE command. The sample report uses the EMPLOYEE data source for its data.
3. The heading section, initiated by the HEADING command, defines the body of the report. Most of the text and data fields that appear in the report are specified in the heading section. In this request, the heading section continues until the FOOTING command is encountered.
4. This line illustrates how versatile you can be with a heading. It shows the following:
  - The second line of the text in the report heading.
  - A data field embedded in the text: <DEPARTMENT.
  - The start position of the line, column 14: <14>.
5. You can enhance the readability of a report using line-skipping commands. The command </2, when coded on a line by itself, generates three blank lines, as seen between the report heading and employee name.
6. This line illustrates how to perform a field calculation in a free-form report using a prefix operator. In this case, we requested the date on which the most recent course was taken—that is, the maximum value for the DATE\_ATTEND field.
7. The next three lines illustrate the use of special characters to create a graphic in the report. The box around EDUCATION CREDITS EARNED may need adjustment for output displayed in a proportional font.
8. The value of the field created by the DEFINE command displays in the box, highlighting the number of education credits an employee has earned. This line demonstrates that you can display a virtual field in the body of your report. As you recall, we created this field at the start of the request.
9. The FOOTING command not only signifies the beginning of the footing section, but, in our example, it also ends the heading section. Since we have designed a personnel report, it is important to have the words PRIVATE AND CONFIDENTIAL appear at the end of each page of the report. The footing can accomplish this requirement.
10. This line illustrates the use of sorting in a free-form report. The report specifications require that information for only one employee appears per page; that requirement is met through the BY and PAGE-BREAK commands.
11. You can specify record selection in a free-form report. As a result of the WHERE criterion, the report includes only employees who have accumulated in-house education credits.



## Designing a Free-Form Report

To design the body of a report, use the **HEADING** and **FOOTING** commands. They enable you to do the following:

- Incorporate text, data fields, and graphic characters in your report.
- Lay out your report by positioning text and data in exact column locations and skipping lines for readability.

Use the **HEADING** command to define the body of a free-form report, and the **FOOTING** command to define what appears at the bottom of each page of a report. A footing in a report is optional. You can easily define an entire report using just a heading.

## Incorporating Text in a Free-Form Report

You can specify text anywhere in a free-form report, for a variety of purposes. In the sample request, text is used:

- As a report title:  
`"<13>EMPLOYEE EDUCATION HOURS REPORT"`
- As a label for data fields:  
`"EMPLOYEE NAME:     <FIRST_NAME> <LAST_NAME>"`
- With a data field and graphic characters:  
`"<10>| EDUCATION CREDITS EARNED <CR_EARNED>| "`
- As a page footing:  
`"<15>PRIVATE AND CONFIDENTIAL"`

## Incorporating Data Fields in a Free-Form Report

The crucial element in any report, free-form or otherwise, is the data. The data fields available for use in a request include data fields in the Master File, cross-referenced fields, and virtual fields created with the DEFINE command.

The sample request for the EMPLOYEE EDUCATION HOURS REPORT, which you can see in *Request for EMPLOYEE EDUCATION HOURS REPORT* on page 17-4 references all three types of data fields:

- ED\_HRS is found in the EMPLOYEE Master File:  
"TOTAL NUMBER OF EDUCATION HOURS: <ED\_HRS>"
- DATE\_ATTEND is found in the EDUCFILE Master File, which is cross-referenced in the EMPLOYEE Master File:  
"MOST RECENT COURSE TAKEN ON: <MAX.DATE\_ATTEND>"
- CR\_EARNED is created with the DEFINE command before the TABLE FILE command and is referenced as follows:  
"<10>| EDUCATION CREDITS EARNED <CR\_EARNED>|"

You can also apply a prefix operator to a data field to select a particular value (for example, the maximum value within a sort group) or to perform a calculation (for example, to compute the average value of a field). You can use any available prefix operator in a free-form report.

In the sample request, the MAX prefix operator selects the most recent completion date of an in-house course:

"MOST RECENT COURSE TAKEN ON: <MAX.DATE\_ATTEND>"

As is true with all types of reports, you must understand the structure of the data source to use the prefix operators correctly.

## Incorporating Graphic Characters in a Free-Form Report

The use of graphics in a report can be as creative as your imagination. The sample report uses special characters to enclose text and a virtual field in a box. Some other ideas include the following:

- Highlighting key data fields using asterisks or other special characters available directly from your keyboard, or using the HEXBYT subroutine. See the *Using Functions* manual for details on HEXBYT.
- Enclosing the entire report in a box to give it a form-like appearance.
- Using double lines to separate the body of the report from its heading and footing.

The use of special characters to create graphics is limited by what can be entered and viewed from your workstation and what can be printed on your printer. If you have any difficulty producing the graphics that you want, be sure to check with someone in your organization who knows what is available.

## Laying Out a Free-Form Report

To provide spacing in a report and position text and data fields as desired, use the spot marker feature of the HEADING and FOOTING commands.

The sample request for the EMPLOYEE EDUCATION HOURS REPORT, which you can see in *Request for EMPLOYEE EDUCATION HOURS REPORT* on page 17-4, illustrates this feature. The first two examples show how to position text and data fields on your report, while the third example shows how to skip lines:

- The spot marker <13> positions the specified text in column 13 of the report:  
`"<13>EMPLOYEE EDUCATION HOURS REPORT"`
- The spot marker <23> positions the specified data field in column 23 of the report:  
`"<23><ADDRESS_LN2>"`
- The spot marker </1 on a line by itself skips two lines after displaying the job description:  
`"JOB DESCRIPTION: <JOB_DESC>"`  
`"</1 "`  
`"MOST RECENT COURSE TAKEN ON: <MAX.DATE_ATTEND>"`

When designing a free-form report, take advantage of sort field options, such as NOPRINT, PAGE-BREAK, and UNDER-LINE. The sample request uses PAGE-BREAK to place each employee's information on a separate page:

```
BY EMP_ID NOPRINT PAGE-BREAK
```

## Sorting and Selecting Records in a Free-Form Report

As with tabular and matrix reports, you can both sort a report and conditionally select records for it. Just use the same commands you would use for tabular and matrix reports. For example, use the BY phrase to sort a report and define WHERE criteria to select records from the data source.

---

## CHAPTER 18

# Creating Graphs: GRAPH

### Topics:

- Introduction
- Command Syntax
- Graph Forms
- Adjusting Graph Elements
- Special Topics
- Special Graphics Devices
- Command and SET Parameter Summary

Graphs often convey meanings more clearly than data listed in tabular report format. The FOCUS GRAPH command acts in the same way as the TABLE command to retrieve data from a file, but it presents the information—either on the screen or to a printer in one of five standard graphic formats:

- A connected point or line plot
- A histogram
- A bar chart
- A pie chart
- A scatter diagram

# Introduction

This chapter explains how to generate each of these graph forms and adjust the features on the graphs you produce.

The examples in this chapter are drawn on the SALES database that is included on your system tape. All examples in this chapter assume that FOCUS default parameters, called SET parameters, are in effect.

The SALES database is used to illustrate the examples used in this chapter. The Master File and a schematic diagram of the file appear in Appendix A, *Master Files and Diagrams*. An additional temporary field named SALES has been defined and used in many of the examples:

```
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END
```

## GRAPH vs. TABLE Requests

GRAPH request syntax is similar to TABLE request syntax. In fact, the output from many TABLE requests can be converted directly into a graph by typing the command REPLOT at the FOCUS command prompt immediately after the output of the request has been displayed. For example

```
TABLE FILE SALES
HEADING CENTER
"SAMPLE TABLE REPORT FOR REPLOTTING"
SUM SALES ACROSS CITY
END
```

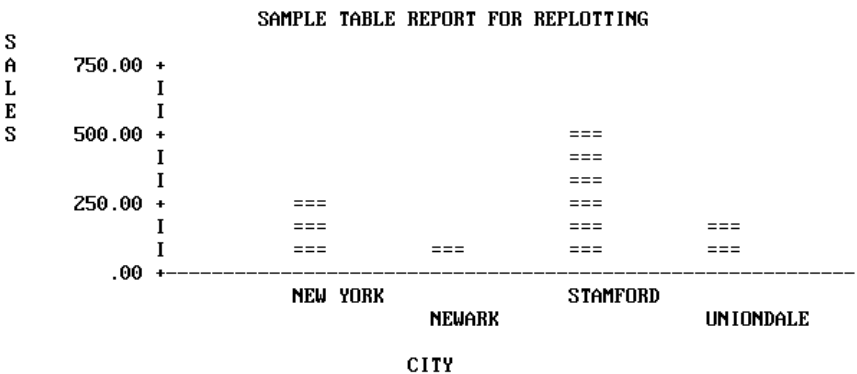
produces the following output:

| SAMPLE TABLE REPORT FOR REPLOTTING |        |          |           |
|------------------------------------|--------|----------|-----------|
| CITY                               |        |          |           |
| NEW YORK                           | NEWARK | STAMFORD | UNIONDALE |
| 224.88                             | 56.08  | 535.34   | 151.85    |

To convert the output into a graph, exit the report and at the FOCUS command prompt type

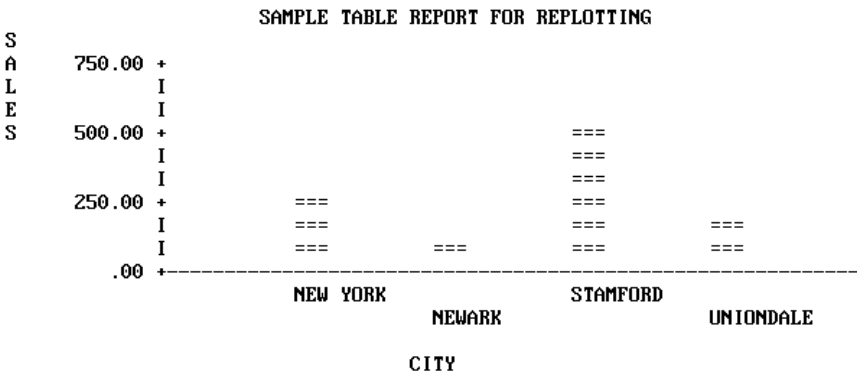
REPLOT

and press Enter.



To produce the graph directly, without creating a preliminary tabular report, substitute the command GRAPH for TABLE in the original request, as shown in the following example:

GRAPH FILE SALES  
HEADING CENTER  
"SAMPLE TABLE REPORT FOR REPLOTTING"  
SUM SALES ACROSS CITY  
END



Thus, you can produce graphs by simply converting TABLE requests, but every TABLE facility does not have a GRAPH counterpart, and there are some practical limitations on the amount of information that you can effectively display in a graph. *Command Syntax* on page 18-12 describes the use of TABLE features in GRAPH requests.

The type of graph (graph form) produced by a GRAPH request depends on the verb used (such as SUM or PRINT), the sort phrase used (ACROSS or BY), and the data type of the sort field. Consider the five graphs that appear on the following pages, and the requests that produce them.

```
SET HISTOGRAM=OFF

GRAPH FILE SALES
HEADING CENTER
"SAMPLE CONNECTED POINT PLOT"
SUM SALES ACROSS DATE
END
```

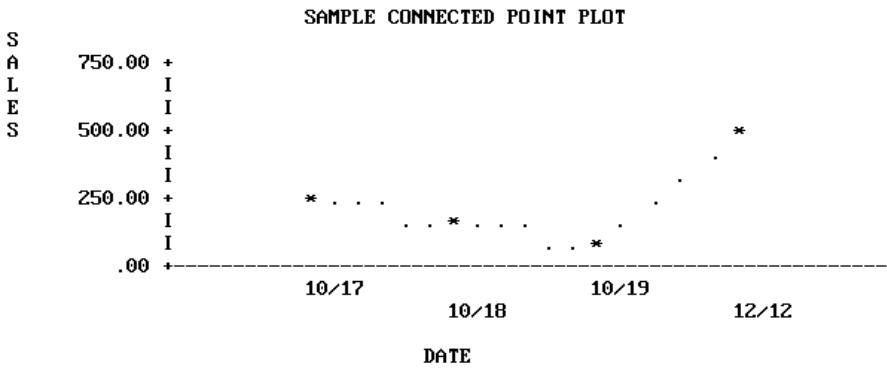


Figure 18-1. Sample Connected Point Plot

**Note:** SET parameters remain in effect until you reset them or log off (see *SET Parameters* on page 18-60).

```
SET HISTOGRAM=ON

GRAPH FILE SALES
HEADING CENTER
"SAMPLE HISTOGRAM"
SUM SALES ACROSS PROD_CODE
END
```

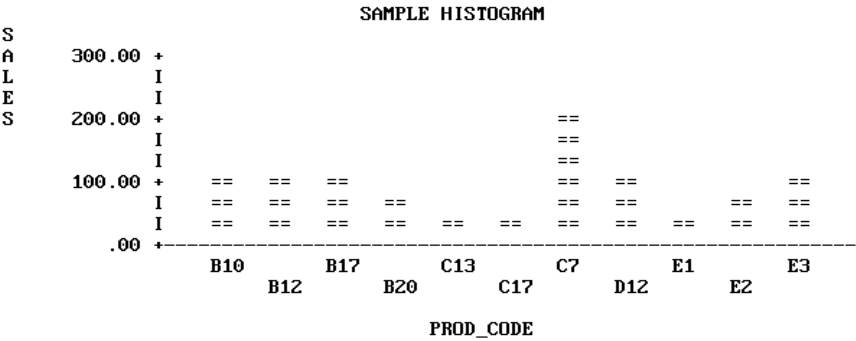


Figure 18-2. Sample Histogram

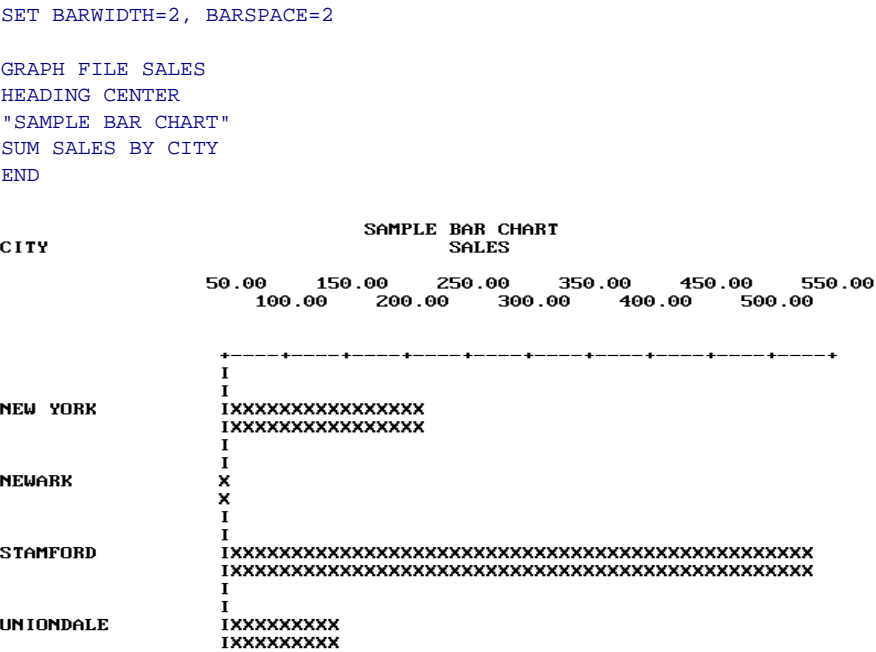


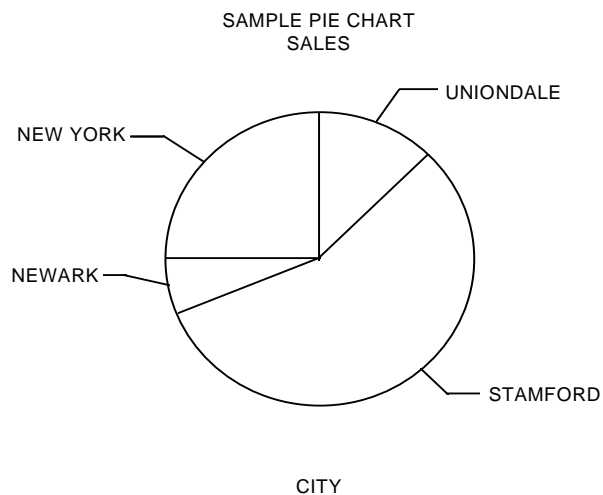
Figure 18-3. Sample Bar Chart



```
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END

SET PIE=ON, GCOLOR=OFF
SET VAXIS=50, HAXIS=100

GRAPH FILE SALES
HEADING CENTER
"SAMPLE PIE CHART"
SUM SALES ACROSS CITY
END
```



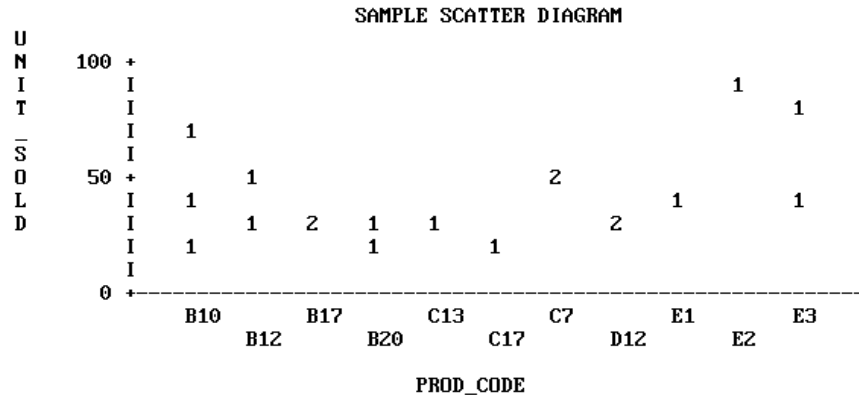
***Figure 18-4. Sample Pie Chart***

```

SET HISTOGRAM=OFF

GRAPH FILE SALES
HEADING CENTER
"SAMPLE SCATTER DIAGRAM"
PRINT UNIT_SOLD ACROSS PROD_CODE
END

```



*Figure 18-5. Sample Scatter Diagram*

## Controlling the Format

In each of the previous graphs, FOCUS created a clear representation of the data using its default values for the graph features (such as axis lengths, axis scales, or titles). You can issue your initial request immediately and concentrate on selecting the data, while FOCUS controls all of the features on the graph.

Subsequently, when satisfied with the data portrayed in your graph, you can refine its appearance by adjusting the parameters that control the look of the graph. You can set the control parameters individually (for example, SET GRID=ON), or ask FOCUS to prompt you for all of their values when you execute the SET GPROMPT=ON command.

**Note:** When entering several SET parameters on one line, separate them with commas.

The request below illustrates some of the parameters you can control:

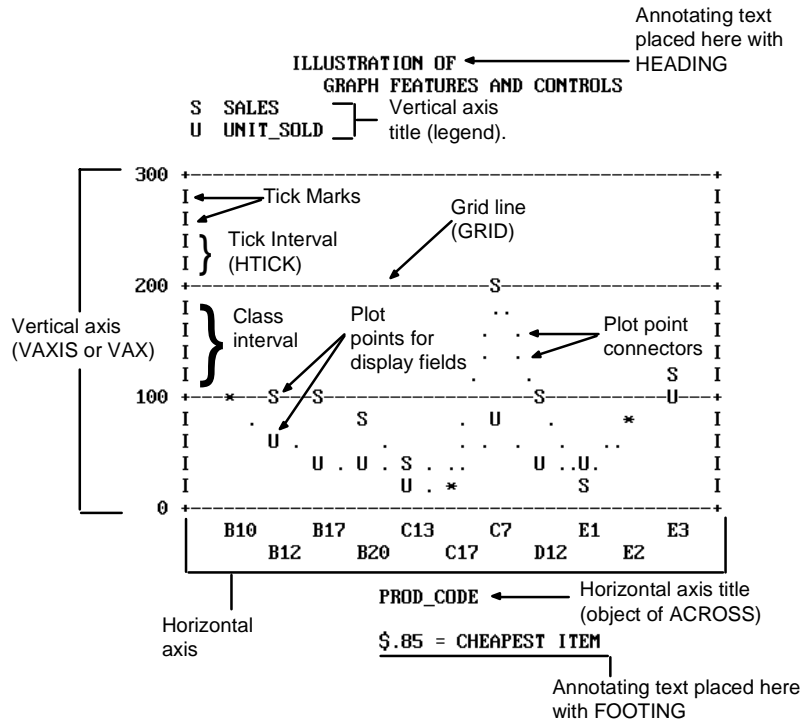
```

SET HISTOGRAM=OFF
SET HAXIS=75, VAXIS=32, GRID=ON

GRAPH FILE SALES
HEADING
"</1 <20 ILLUSTRATION OF"
"<23 GRAPH FEATURES AND CONTROLS"
SUM SALES AND UNIT_SOLD ACROSS PROD_CODE
FOOTING CENTER
"</1 <MIN.RETAIL = CHEAPEST ITEM"
END

```

The graph generated OFFLINE in response to the request appears below.



**Figure 18-6. Illustration of Graph Elements**

**Note:**

- This graph is a connected point plot, with the plot points representing the sales (retail\_price \* unit\_sold) and total units sold for each of the product codes listed across the horizontal axis.
- Annotating text has been added above and below the graph with the HEADING and FOOTING facilities. Note the use of spot markers to position text on the graph and the embedded calculation with a direct operator.
- Only the vertical axis is scaled because the ACROSS phrase objects were not numeric values. The plus symbols (+) mark the class intervals on the axis scale and vertical bars mark the tick intervals.
- Horizontal grid lines appear at the vertical class marks.

For graphs generated ONLINE, FOCUS automatically detects the height and width of a particular terminal and plots the graph accordingly. As a result, VAXIS and HAXIS settings are ignored.

You control the graphic elements shown in Figure 18-6 in one of two ways: either by the syntax in the actual request, or with SET commands. *Command Syntax* on page 18-12 describes the elements in GRAPH requests and their effects. *Adjusting Graph Elements* on page 18-38 describes the adjustable parameters that control graph features.

There are also some additional SET parameters that control non-graphic elements:

- Specifying an output device.
- Pausing between data retrieval and printing to permit the user to adjust paper in the printer or plotter.
- Using special black/white shading patterns to simulate different colors.
- Displaying the current settings of the GRAPH parameters on the screen.

After retrieving data from a file and displaying it either as a tabular report or a graph, you can use the SET command to adjust the format and then redisplay the graph by issuing the REPLOT command (without resorting to further data retrieval).

A summary of all of the SET parameters appears in *SET Parameters* on page 18-60.

# Graphic Devices Supported

You may create graphs on any terminal or printer that can print FOCUS reports. If your terminal has no graphics capabilities, FOCUS uses the characters in the standard character set when producing graphs. As the default, FOCUS sends GRAPH output to the terminal (or system printer, if PRINT=OFFLINE). This produces low-resolution graphics. The examples in the chapter thus far (except the pie chart) illustrate the default. You cannot create continuous line plots or pie charts unless you have a high-resolution graphic device.

While FOCUS can accommodate devices with no inherent graphics capabilities, it can also take advantage of whatever graphics facilities are available. Some personal computers offer ranges of special characters that can be used to create more readable graphs (see Figure 18-7), and if color monitors or multiple-pen plotters are available, further improvements in graph quality are possible (see Figure 18-8 and *Special Graphics Devices* on page 18-54).

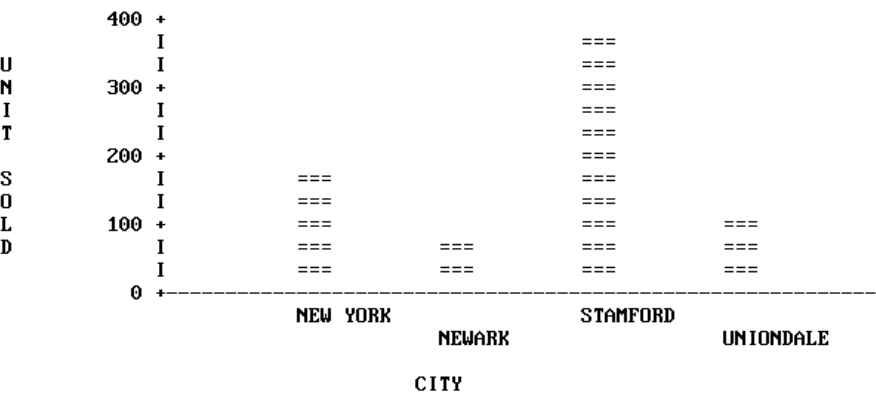


Figure 18-7. Graph on an IBM PC Mono Screen

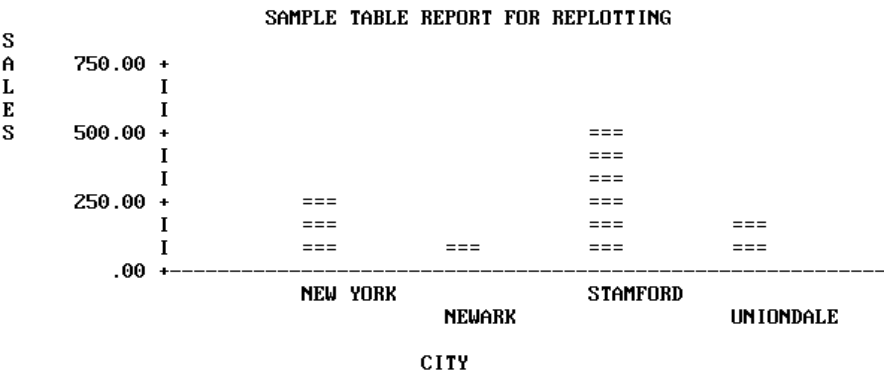


Figure 18-8. Sample Graph on Plotter

On IBM mainframes, FOCUS supports the use of high-resolution terminals such as the Model 3279 via the IBM Graphical Data Display Manager (GDDM), which is discussed in *IBM Devices Using GDDM* on page 18-55. A variety of other high-resolution terminals, printers, and plotters are also supported and they are listed in this section. To select one, simply enter the appropriate form of the SET DEVICE command (see *High-Resolution Devices* on page 18-55). Note, in reviewing the device selections, that all have fixed graphic window dimensions (horizontal and vertical axes), which are fixed until a new device is selected.

Please note that this list includes only fully tested devices, although other devices may also work with FOCUS.

## Medium-Resolution Devices

Anderson Jacobson Models: AJ830, and AJ832 (12 Pitch).

Diablo Models: 1620, and 1620 (12 Pitch).

Gencom Models: GENCOM, and GENCOM (12 Pitch).

Trendata Models: Trendata 4000A, and 4000A (12 Pitch).

## High-Resolution Devices

IBM Graphic Devices (GDDM is required).

- Any IBM 3270 series device that supports GDDM graphics, such as 3279-S3G, 3179, or 3472. This includes PCs with fully compatible 3270 series hardware and software.

Hewlett-Packard Plotters:

- Four-pen plotters without paper advance: Models 7220A, 7221, and 7470A (requires Y cable #17455).
- Four-pen plotters with paper advance: Models 7220S and 7221S.
- Eight-pen plotters without paper advance: Models 7220C, 7221C, 7475A (requires Y cable #17455).
- Eight-pen plotters with paper advance: Models 7220T, 7221T.

Tektronix Graphic Devices (only monochrome display).

- Models 4010, 4012, 4013, 4014, 4014E, 4015, 4015E, 4025, 4027, 4050 series, 4662, and 4100 series.

## Command Syntax

Most TABLE requests can be converted into GRAPH requests by simply replacing the TABLE command with the GRAPH command. The only limitations are those inherent in the nature of the graphic format. When a TABLE request is converted in this manner, the various phrases that make up the body of the request take on special meanings that determine the format and layout of the graph.

This section outlines the phrases that can appear in TABLE requests and describes their effects in the context of GRAPH requests. The section also describes any limitations that apply to their use.

## GRAPH vs. TABLE Syntax

The syntax of the GRAPH command parallels that of the TABLE command. The main elements of GRAPH requests are the verb phrase (display command), one or more sort phrases, selection phrases, and headings and footings. All of the other phrases that appear in TABLE requests are ignored. This applies to all control conditions (ON...) and all IN phrases. The basic GRAPH syntax is as follows:

```
GRAPH FILE filename
[HEADING]
[heading phrase]
verb phrase
sort phrase
[additional sort phrases]
[selection phrase(s)]
[FOOTING]
[footing phrase]
END
```

The GRAPH request elements generally follow the same rules as their TABLE counterparts:

- The word FILE and the file name must immediately follow the GRAPH command, unless they were previously specified in a SET command:

```
SET FILE=filename
```

The file named can be any file available to FOCUS, including joined or cross-referenced structures.

- You can concatenate unlike data sources in a GRAPH request with the MORE command. See *Concatenating Unlike Data Sources* on page 18-17.

- The order of the phrases in the request does not affect the format of the graph. For example, the selection phrase may follow or precede the verb phrase and sort phrase(s). The order of the sort phrases does affect the format of the graph, however, just as the order of the sort phrases in TABLE requests affects the appearance of the reports (see *Selecting Forms: BY and ACROSS Phrases* on page 18-15).
- The word END must be typed on a line by itself to complete a GRAPH request.
- An incomplete GRAPH request can be terminated by typing the word QUIT on a line by itself (instead of END).
- All dates are displayed in MDY format unless they are changed to alphanumeric fields.

There are a few notable syntactical differences between TABLE and GRAPH. Specifically, the following restrictions apply:

- GRAPH requests must contain at least one sort phrase (BY phrase or ACROSS phrase) and a verb with at least one verb object in order to generate a meaningful graph.
- Several BY phrases can be used in a request, in which case multiple graphs are created (one for each BY object). A single ACROSS phrase is allowed in a GRAPH request, and requests for certain graph forms can contain both ACROSS and BY phrases.
- The number of ACROSS values cannot exceed 64.
- In GRAPH requests the verb object must always be a numeric field.
- In GRAPH requests the sortfield in ACROSS and BY phrases cannot be a date format field.
- No more than five verb objects are permitted in a GRAPH request. (This limitation is necessary because standard graph formats generally do not permit more variables to be displayed without rendering the graph unreadable.)
- The RUN option is not available as an alternative to END.

The following sections describe in more detail the functions performed by each of the phrases used in GRAPH requests.



## Specifying Graph Forms and Contents

Each graph form is defined by a particular combination of verb and sort phrase. The combinations, which were illustrated earlier in *GRAPH vs. TABLE Requests* on page 18-2, are summarized in the table below (A and B represent two field names).

```
Point plot: SUM A ACROSS B (B is numeric)
Histogram: SUM A ACROSS B (B is alpha) requires SET HISTOGRAM=ON
Bar chart: SUM A BY B
Pie chart: SET PIE=ON
 SUM A ACROSS B
Scatter diagram: PRINT A ACROSS B or PRINT A BY B
```

**Figure 18-9. Graph Selection Phrases**

### Naming Subjects: Verb Phrases

Each GRAPH request must include a verb and at least one verb object (up to five are allowed). Three verbs are permitted: COUNT, SUM, and PRINT. SUM is synonymous with either WRITE or ADD. Each verb object must be a computational field (not alphabetic). For example,

```
GRAPH FILE SALES
SUM SALES
.
.
.
```

If the verb SUM (or WRITE or ADD) is used, then a bar chart, histogram, line plot or pie chart is produced, depending on the sort phrase and sort field format used. If PRINT is used, the graph is a scatter diagram.

The verb objects, which are the subjects of the graph, may be real or defined fields, with or without direct operation prefixes (AVE., MIN., MAX., etc.). They may also be computed fields. (All of the COMPUTE facilities are available in GRAPH requests.)

When the request has a single verb object, the vertical title of the graph is either the field name of the verb object as it appears in the Master File or a replacement name supplied in an AS phrase.

When a request contains multiple verb objects, each represents one variable in the graph, and a vertical legend is printed instead of the vertical title. The legend specifies the field names (and/or AS phrase substitutions) and provides a key to which line represents each variable.

In your requests, verb objects may be separated by spaces, or by AND or OVER. OVER has special significance in histogram and bar chart requests, where it controls the stacking of the bars. This is described in the sections on Histograms (see *Histograms* on page 18-26), and Bar Charts (see *Bar Charts* on page 18-29).

Verb objects used only for calculations need not appear in your graphs. Use the NOPRINT or SUP-PRINT facilities to suppress the display of such fields.

## Selecting Forms: BY and ACROSS Phrases

At least one sort phrase is required in every GRAPH request. This may be either a BY phrase or an ACROSS phrase.

For example,

```
GRAPH FILE SALES
SUM SALES
ACROSS PROD_CODE
.
.
.
```

The ACROSS phrase, if there is one, determines the horizontal axis of the graph.

If there is no ACROSS phrase the last BY phrase determines the vertical axis. When there are multiple BY phrases or when an ACROSS and BY phrase are included in the same request, FOCUS generates multiple graphs; one for each combination of values for the fields referenced in the request (see *The Vertical Axis: System Defaults* on page 18-43 for information regarding control of the vertical axis).

**Note:** The FOCUS ICU Interface saves data for IBM's Interactive Chart Utility (ICU) in tied data format. If both an ACROSS and BY phrase are present in a GRAPH request one common axis is established. This enables FOCUS graphs to be displayed as tower charts.

The FOCUS ICU Interface is discussed in further detail in *Using the FOCUS ICU Interface* on page 18-54. You can also consult the *ICU Interface Users Manual* for additional information.

The sortfield name may be replaced with an AS phrase. This is useful if the sort phrase specifies one of the axes (it has no effect on any additional sort phrases).

Note that the values of fields mentioned in the additional sort phrases are not displayed automatically in the graph. If you wish to have them appear you must embed them in a heading or a footing line (see *Adding Annotating Text: HEADING and FOOTING Lines* on page 18-18).

### Selecting the Contents: Selection Phrases

Selection phrases are used in GRAPH requests to select particular records of interest. Two selection phrases are available: IF and WHERE. The examples in this chapter use the IF selection phrase. For a definition of the WHERE clause and the differences between IF and WHERE, see Chapter 5, *Selecting Records for Your Report*.

The syntax for an IF phrase or a WHERE clause in a GRAPH request is identical to that used in a TABLE request. For example,

```
GRAPH FILE SALES
SUM SALES
ACROSS PROD_CODE
IF PROD_CODE NE D12
```

A partial list of the relation tests appears below. See Chapter 5, *Selecting Records for Your Report*, for a complete list.

| Relation | Meaning                  |
|----------|--------------------------|
| EQ       | Equal to                 |
| NE       | Not equal to             |
| GE       | Greater than or equal to |
| GT       | Greater than             |
| LE       | Less than or equal to    |
| LT       | Less than                |
| CONTAINS | Contains                 |
| OMITS    | Omits                    |

## Concatenating Unlike Data Sources

With the FOCUS command, MORE, you can graph data from unlike data sources in a single request; all data, regardless of source, appears to come from a single file. You must divide your request into:

- One main request that retrieves the first file and defines the data fields, sorting criteria, and output format for all data.
- Subrequests that define the files and fields to be concatenated to the data of the main request. The fields printed and sorted by the main request must exist in each concatenated file. If they do not, you must create them as DEFINE fields.

During retrieval, FOCUS gathers data from each database in turn. It then sorts all data and formats the output as described in the main request. The syntax is

```
GRAPH FILE file1
 main request
MORE
FILE file2
 subrequest
 MORE
 .
 .
 .
END
```

where:

*file1*

Is the name of the first file.

*main request*

Is a request, without END, that describes the sorting, formatting, aggregation, and COMPUTE field definitions for all data. IF and WHERE phrases in the main request apply only to *file1*.

MORE

Begins a subrequest. The number of subrequests is limited only by available memory.

FILE *file2*

Defines *file2* as the second file for concatenation.

*subrequest*

Is a subrequest. Subrequests can only include WHERE and IF phrases.

END

Ends the request.

See Chapter 14, *Merging Data Sources*, for complete information and for concatenation examples.

## Adding Annotating Text: HEADING and FOOTING Lines

To insert annotating text above or below a graph, enter the keywords **HEADING** and/or **FOOTING**, followed by the desired contents, including any necessary control elements for skipping lines, etc. The syntax is the same as that used for headings and footings in **TABLE** requests.

For example,

```
GRAPH FILE SALES
HEADING
"<7 THIS GRAPH SHOWS SALES BY PRODUCT CODE"
SUM SALES
BY PROD_CODE
IF PROD_CODE NE D12
FOOTING
"<7 FOR ALL PRODUCT CODES EXCEPT D12"
END
```

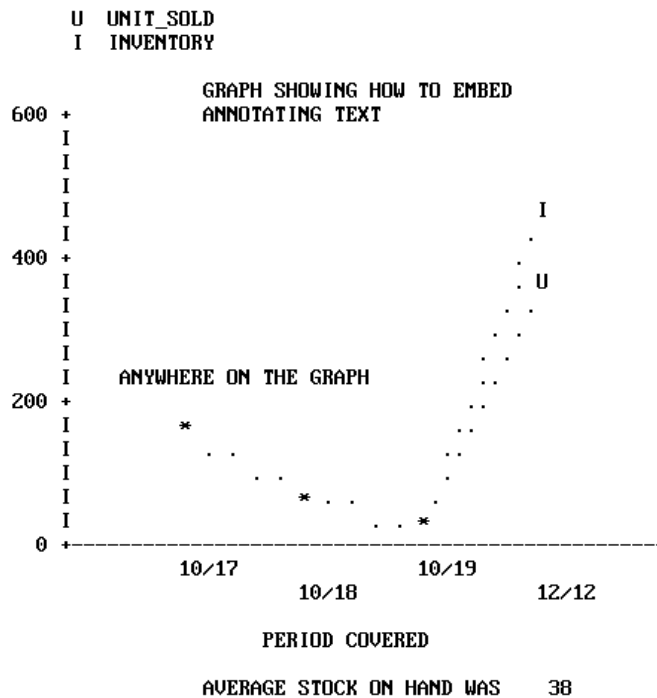
**Note:** When annotating text falls in the path of a plot point on a graph, the plot point is printed; however, connecting points are suppressed if they lie in the path of annotating text. (This enables you to adjust the position of the annotating text when you see the contents of the graph.) The first line of any heading appears above the first line of the legend.

## Inserting Formatting Controls

The formatting controls used in TABLE requests can also be used in GRAPH requests for positioning text or field references in heading or footing lines, or even in the body of your graph. The following example shows the use of spot markers, which are described in

Chapter 9, *Customizing Tabular Reports*.

```
SET HISTOGRAM=OFF
GRAPH FILE SALES
HEADING
"</4 <22 GRAPH SHOWING HOW TO EMBED"
"<22 ANNOTATING TEXT"
"</10 <15 ANYWHERE ON THE GRAPH"
SUM UNIT_SOLD AND OPENING_AMT AS 'INVENTORY'
ACROSS DATE AS ' PERIOD COVERED'
FOOTING CENTER
"AVERAGE STOCK ON HAND WAS <AVE.OPENING_AMT"
END
```

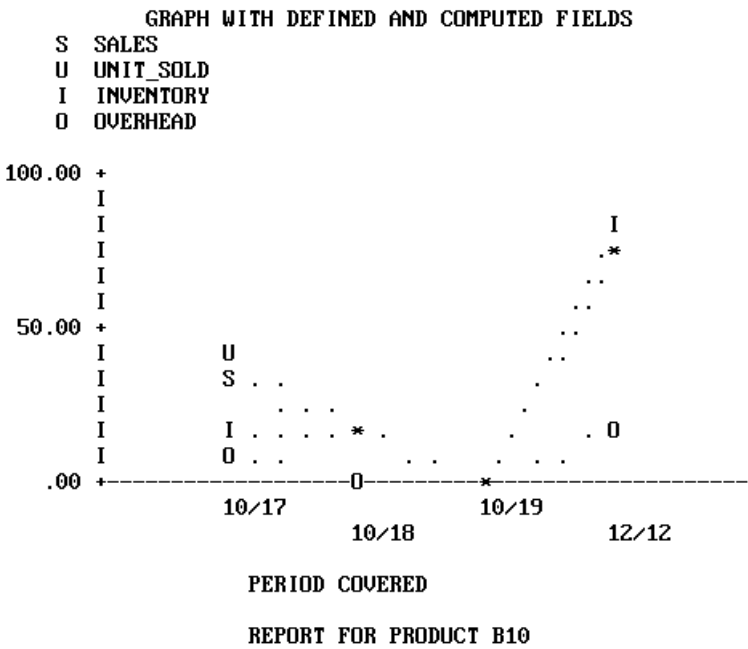


## Inserting Field References

The following example shows how to embed field values in graph heading or footing lines. This capability is analogous to that in TABLE requests, and is useful when annotating graphs created by requests containing multiple sort fields (where only the first named sort field will appear as a title on the graph).

```
SET HISTOGRAM=OFF
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END

GRAPH FILE SALES
HEADING CENTER
"GRAPH WITH DEFINED AND COMPUTED FIELDS"
SUM SALES AND UNIT_SOLD AND OPENING_AMT
AS 'INVENTORY' AND
COMPUTE OVERHEAD/D8.2=.20 * SALES;
ACROSS DATE AS ' PERIOD COVERED'
BY PROD_CODE
IF 'PROD_CODE' IS 'C7' OR 'B10' OR 'B12'
FOOTING CENTER
"REPORT FOR PRODUCT <PROD_CODE>"
END
```



# Graph Forms

This section describes the five graph forms produced by FOCUS, and their basic elements. Connected point plots are described first, followed by histograms, bar charts, pie charts, and scatter diagrams. The adjustable graphic features are mentioned only briefly with the graph forms and fully described in *Adjusting Graph Elements* on page 18-38.

As you may have noticed when viewing the examples in *GRAPH vs. TABLE Requests* on page 18-2, there are similarities between the requests used to create some of the forms. For example, a request for a connected point plot (with an alphanumeric ACROSS field) will create a histogram instead if the HISTOGRAM parameter is set on (the default). This feature enables you retrieve data once, and then flip back and forth from one form to the other by changing the HISTOGRAM value and issuing REPLOT.

Histograms are often called vertical bar charts, but the physical similarities between these forms mislead users. Although the graphs look similar and have parameters that perform similar functions (HSTACK and BSTACK), the parameters used to control the widths and spacing of bars on bar charts have no effect on histogram bars.

Histograms and vertical scatter plots (those created with BY phrases) have variable-length vertical axes that are not subject to the VAXIS parameter setting (VAXIS is ignored).

Pie charts and bar charts are different geometrical representations of similar types of data, but pie charts are only possible if you have a high-resolution device capable of drawing respectable curves.



# Connected Point Plots

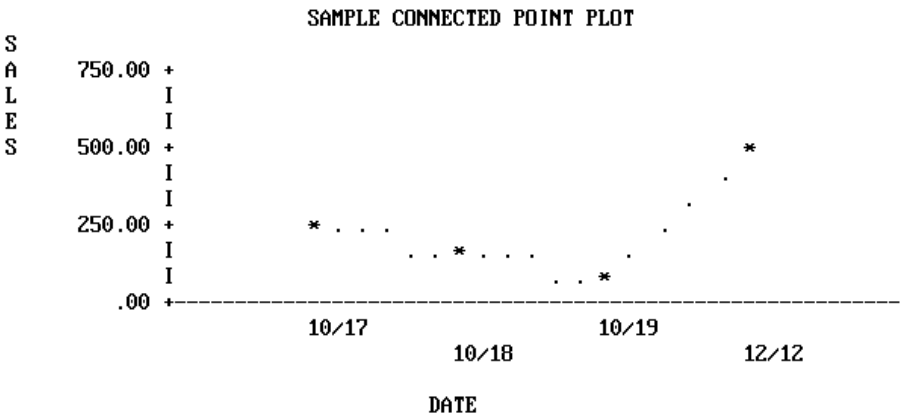
You create a connected point plot (or a line plot on a high-resolution device), with a request that combines the verb SUM (or the synonyms WRITE or ADD) with an ACROSS phrase that specifies an alphanumeric or a numeric field. If the field specified in the ACROSS phrase is alphanumeric, the HISTOGRAM parameter must be set off in order to generate a connected point plot.

The values for the field named in the ACROSS phrase are plotted on the horizontal axis and the values for the verb object(s) are plotted along the vertical axis.

The example below illustrates a point plot request.

```
SET HISTOGRAM=OFF

SET VAXIS=40,HAXIS=75
GRAPH FILE SALES
HEADING CENTER
"SAMPLE CONNECTED POINT PLOT"
SUM SALES ACROSS DATE
END
```



**Note:** The SET statements in the previous example were added to limit the size of the output graph to a convenient size for display on the page. Without them, FOCUS sets the default horizontal axis width at the capacity of the device selected, and a vertical height of 66 lines (normal page length).

## Point Plot Features

**Scale Titles** — The values associated with the class markers are printed below the horizontal axis in the USAGE format of the variable being plotted (MM/DD in our example).

**Plot Characters** — The graphics characters used to plot the variables on connected point plots vary depending on the type of display device being used:

- On high-speed printers and non-graphics terminals the data points are represented by asterisks (\*) when only one variable is plotted. If several variables are plotted, the initial letters of the variable names are used (if you have duplicates, rename them with AS phrases). The data points are connected by periods(.). You cannot create continuous line plots on these devices; they are only available on high-resolution devices.
- On high-resolution displays, printers, and plotters the lines connecting plot points are drawn explicitly and when there are several variables they are distinguished either by color or by the type of connecting line used (dotted, solid, or broken).

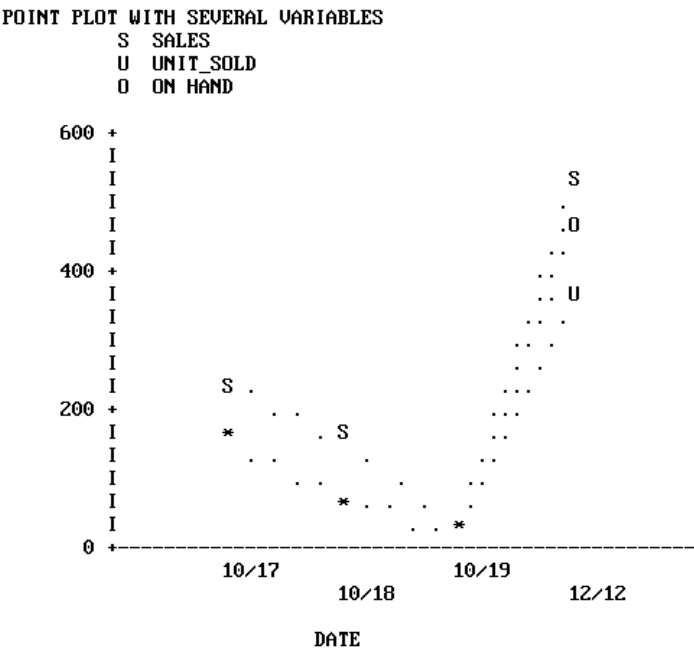
**Axis Titles** — You can include vertical and horizontal axis titles for your graphs:

- For requests with a single verb object the vertical title is either its field name or a replacement name you have provided in an AS phrase.
- When more than one variable is plotted, FOCUS prints a vertical legend instead of the vertical title. The legend specifies the field names or their replacements, and provides a key showing which line represents each variable. Titles are displayed staggered or folded on successive horizontal lines to permit more titles than a single horizontal line can contain.

The example that follows illustrates a point plot with several variables.

```
SET HISTOGRAM=OFF

GRAPH FILE SALES
HEADING
"POINT PLOT WITH SEVERAL VARIABLES"
SUM SALES AND UNIT_SOLD AND INV AS 'ON HAND'
ACROSS DATE
END
```



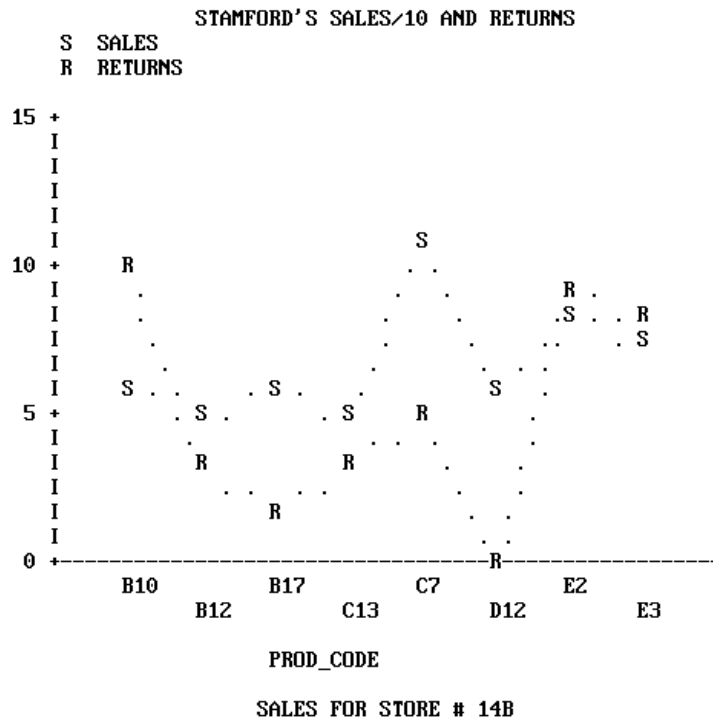
Up to five variables can be plotted on the same vertical axis. When this is done, the scale on the vertical axis is determined based on the combined values of the vertical variables, and a separate point appears for each value of each variable.

When planning graphs with multiple variables or large numbers, adjust your variables so they are in the same order of magnitude. By redefining the variable plotted on the horizontal axis by a suitable power of 10 you can improve the readability of the finished graph. A method for doing this is shown in the example below.

```
DEFINE FILE SALES
SALES/D8.2=(UNIT_SOLD * RETAIL_PRICE)/10;
END
```

```
SET HISTOGRAM=OFF
```

```
GRAPH FILE SALES
HEADING CENTER
"STAMFORD'S SALES/10 AND RETURNS"
SUM SALES AND RETURNS ACROSS PROD_CODE
BY STORE
IF CITY IS 'STAMFORD'
FOOTING CENTER
"SALES FOR STORE # <STORE_CODE>"
END
```

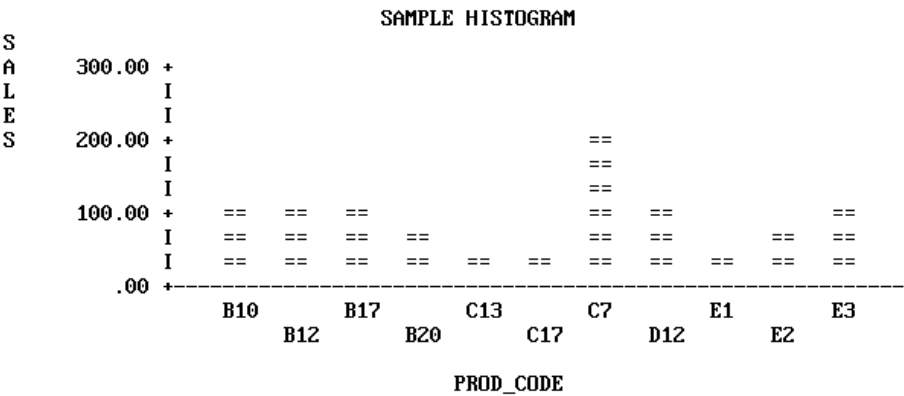


# Histograms

Histograms are vertical bar charts and are useful for portraying the component parts of aggregate values. They are essentially an alternate graphic format for plotting requests that could also generate connected point plots. To flip back and forth from one format to the other, simply reset the parameter HIST and issue REPLOT.

You create histograms by typing requests containing the verb SUM (or the synonyms, WRITE or ADD) and an ACROSS phrase that specifies an alphanumeric field. One bar appears on the graph for each verb object. The example that follows illustrates a histogram with a single variable.

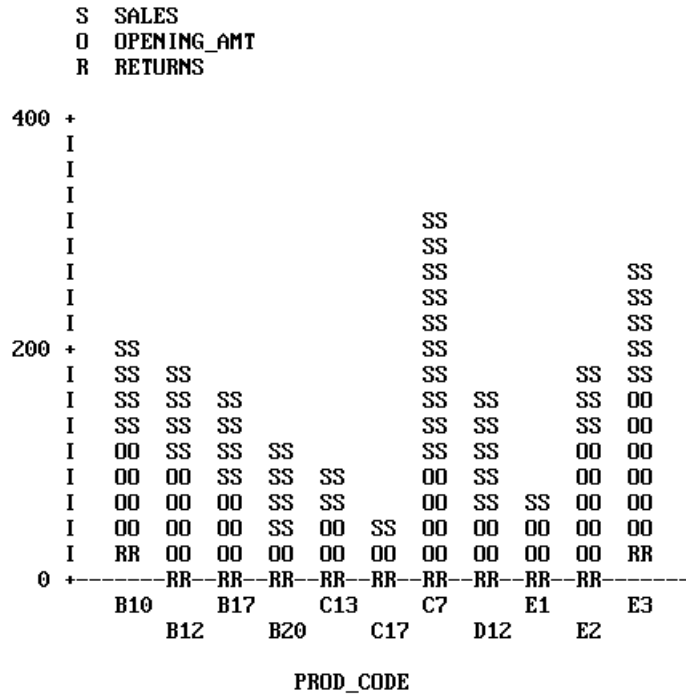
```
GRAPH FILE SALES
HEADING CENTER
"SAMPLE HISTOGRAM"
SUM SALES ACROSS PROD_CODE
END
```



To draw the bars side by side, separate the verb objects with spaces or AND. To draw superimposed (stacked) bars, separate the verb objects with OVER. The example that follows illustrates a request using OVER:

```
GRAPH FILE SALES
HEADING
"SALES OVER INVENTORY AND RETURNS"
"ACROSS PRODUCT CODE"
SUM SALES OVER INV OVER RETURNS ACROSS PROD_CODE
END
```

**SALES OVER INVENTORY AND RETURNS  
ACROSS PRODUCT CODE**



Note that the legend uses the full field names rather than the aliases for the verb objects (OPENING\_AMT for INV).

When you name three or more verb objects in a request, you can have any combination of stacked and side-by-side bars.

## Histogram Features

Each vertical bar or group of bars represents a value of the ACROSS sort field. The range of values for the verb objects determines the scale for the vertical axis.

All of the vertical axis features on histograms are adjustable:

- To reset the height of OFFLINE graphs, use the VAXIS parameter as described in *How to Set the Height* on page 18-43. (For online graphs, FOCUS automatically sets the height of your graph based on the terminal dimensions.)
- You can reset the upper and lower thresholds on the axis by setting the default scaling mechanism off (VAUTO) and setting new upper and lower limits (VMAX and VMIN). See *How to Set the Scale: Assigning Fixed Limits* on page 18-41.
- You can reset the class and tick intervals by overriding the default mechanism (AUTOTICK) and setting new intervals (VCLASS and VTICK). See *How to Set Class and Tick Intervals* on page 18-42.

FOCUS automatically sets the width of the bars and the spacing between them to fit within the HAXIS parameter limit. These can be changed by resetting the HAXIS parameter (see *How to Set the Width* on page 18-40).

The values for the data points on the HAXIS are printed horizontally on a single line or staggered (folded) on two or more lines, depending on the available space.

To add a grid of parallel horizontal lines at the vertical class marks, issue the following SET command before issuing your request:

```
SET GRID=ON
```

(Vertical grids are not available on histograms.)

To specify stacking of all bars without using OVER in the request, you can set the parameter HSTACK (SET HSTACK=ON). (Remember to set it off again before moving to other requests.)

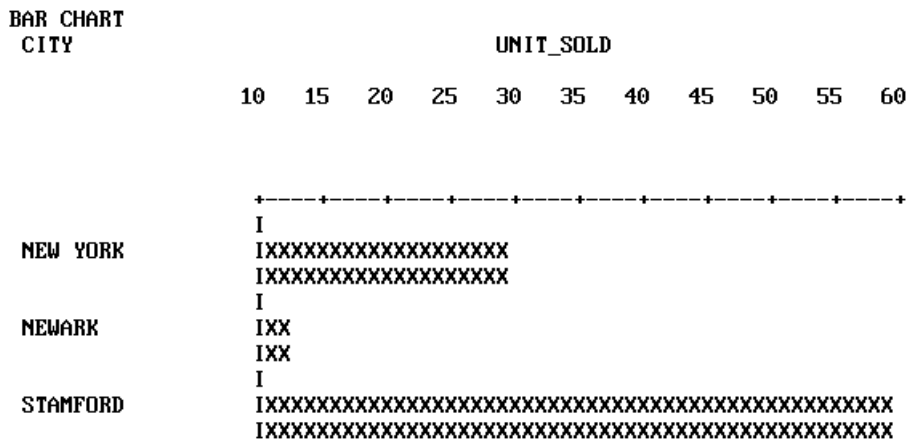
**Note:** There is often confusion over histogram features because of the similarity with bar charts. The BARNUMB facility used to print summary numbers for the bars in bar charts does not work with histograms.

## Bar Charts

Bar charts have horizontal bars arrayed vertically. To produce a bar chart, type a request containing the verb SUM and a BY phrase (but no ACROSS phrase). A separate group of bars is created for each value of the BY field, and each group contains one bar for each verb object in the request.

```
SET BARWIDTH=2, BARSPACE=1
```

```
GRAPH FILE SALES
HEADING
"BAR CHART"
SUM UNIT_SOLD BY CITY
IF PROD_CODE EQ B10
END
```



In the request above, the parameters BARSPACE and BARWIDTH were set to enhance the appearance of the graph and improve readability.



In requests with multiple verb objects, each bar appears beneath its predecessor by default. If verb objects are connected by OVER phrases, however, then the corresponding bars are stacked and appear end-to-end. The following example illustrates stacked bars.

```
SET BARSPACE=2, BARWIDTH=2
```

```
GRAPH FILE SALES
HEADING
"BAR CHART"
SUM DELIVER_AMT OVER INV BY CITY
WHERE 'PROD_CODE' EQ 'B10'
END
```

### BAR CHART

|   |             |
|---|-------------|
| D | DELIVER_AMT |
| O | OPENING_AMT |

CITY

0      20      40      60      80      100      120      140      160

NEW YORK

NEWARK

STAMFORD

Alternatively, to request stacking of all bars, set the parameter **BSTACK** (**SET BSTACK=ON**). If you use **BSTACK** you do not need **OVER**; any graph can be replotted with and without stacking by simply changing the value of this parameter and issuing **RELOT**.

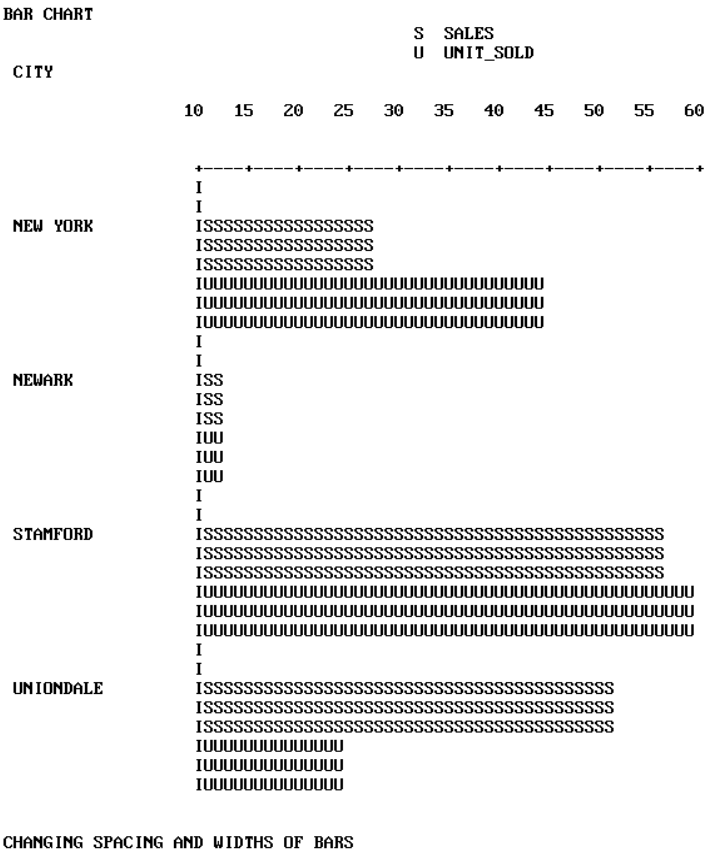
## Bar Chart Features

As we mentioned previously, you can set the BARWIDTH parameter to change the widths of the bars themselves and set the BARSPACE parameter to change the spacing between them. Set the GRID parameter to add a grid of vertical parallel lines at the class marks on the horizontal axis. The examples that follow illustrate the use of these parameters.

```
SET BARWIDTH=3, BARSPACE=2, BSTACK=OFF

GRAPH FILE SALES
HEADING
"BAR CHART"
SUM AVE.SALES AND UNIT_SOLD BY CITY
WHERE 'PROD_CODE' IS 'B10' OR 'B20'
FOOTING
"</2 CHANGING SPACING AND WIDTHS OF BARS"
END
```

The result appears below.

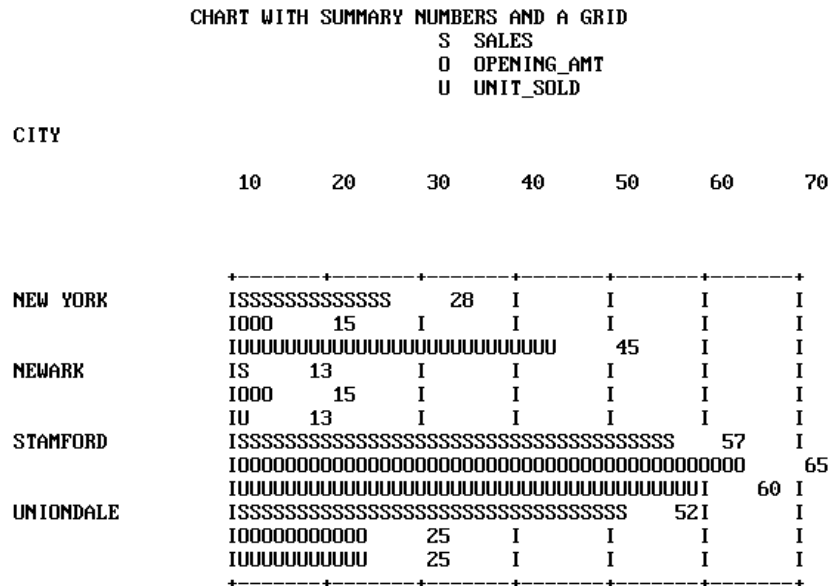


To print a summary value at the end of each bar, set the `BARNUMB` parameter.

**Note:** This feature is also available on pie charts, but is not available on histograms.

The effects of BARNUMB and GRID are shown below.

```
SET BARNUMB=ON, GRID=ON
GRAPH FILE SALES
HEADING CENTER
"CHART WITH SUMMARY NUMBERS AND A GRID"
SUM AVE.SALES AND INV AND UNIT_SOLD BY CITY
WHERE 'PROD_CODE' EQ 'B10' OR 'B20'
END
```



The horizontal axis features are all adjustable:

- To change the width of OFFLINE graphs, alter the HAXIS parameter as described in *How to Set the Width* on page 18-40. (For ONLINE graphs, FOCUS will automatically detect the width of the terminal and display the graph accordingly.)
- To reset the numerical scale, turn off the default scaling mechanism (HAUTO) and set new upper and lower limits (HMAX and HMIN). See *How to Set the Scale: Assigning Fixed Limits* on page 18-43.
- To change the class and tick intervals, override the default mechanism (AUTOTICK) and set new intervals (HCLASS and HTICK). See *How to Set Class and Tick Intervals* on page 18-42.

The vertical axis length is controlled by FOCUS. You can set the bar widths and spacing as mentioned previously, but you cannot set the vertical height to a fixed dimension.

## Pie Charts

Pie charts can only be drawn on high-resolution graphic devices. (It is possible, however, to create a formatted pie chart and save it for subsequent plotting on another device. See *Saving Formatted GRAPH Output* on page 18-51.)

To create a pie chart, first set the PIE parameter ON and select a device (SET DEVICE=), then type a request with the verb SUM (or the synonyms, WRITE or ADD) and an ACROSS phrase that names an alphanumeric field. When you finish your pie charts, set the PIE parameter OFF before running other types of GRAPH requests.

```
SET PIE=ON, DEVICE=HP7220C
```

```
GRAPH FILE SALES
```

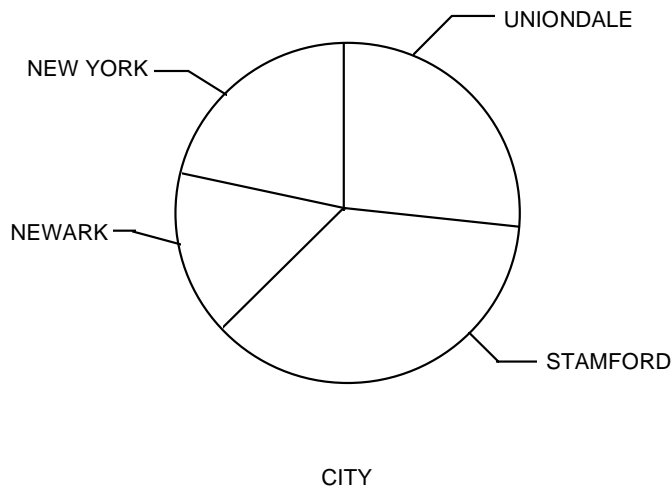
```
HEADING CENTER
```

```
"PIE CHART PRODUCED ON HEWLETT-PACKARD MODEL 7475"
```

```
WRITE RPCT.UNIT_SOLD ACROSS CITY
```

```
END
```

PIE CHART produced on Hewlett-Packard Model 7475  
UNITS\_SOLD

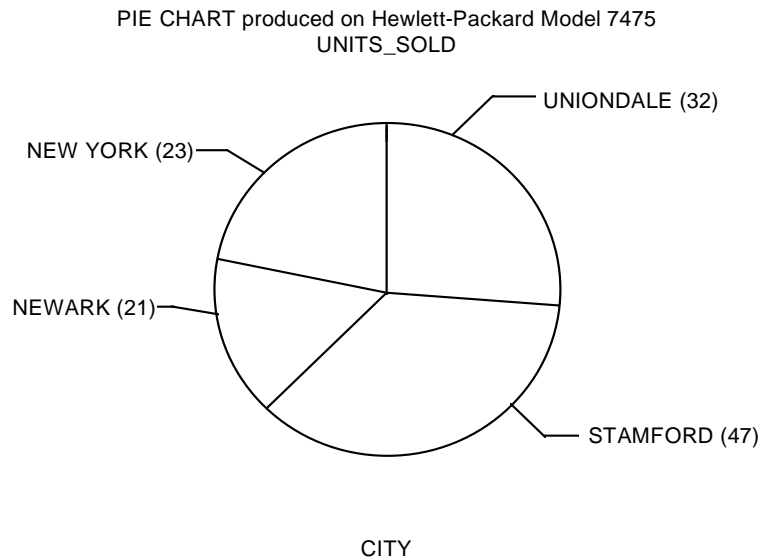


## Pie Chart Features

To add summary numbers for each slice of the pie chart on the previous page, simply enter the following commands:

```
SET BARNUMB=ON
REPLOTT
```

The effect is shown below:

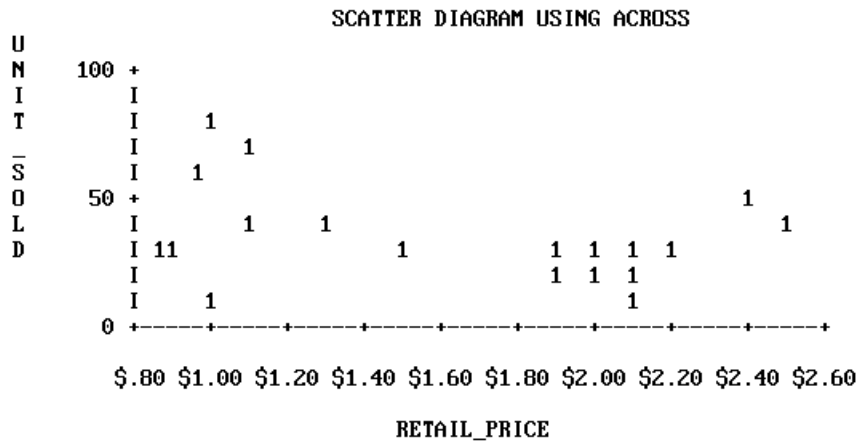


**Note:** FOCUS does not include a facility for displaying exploded pie chart slices.

## Scatter Diagrams

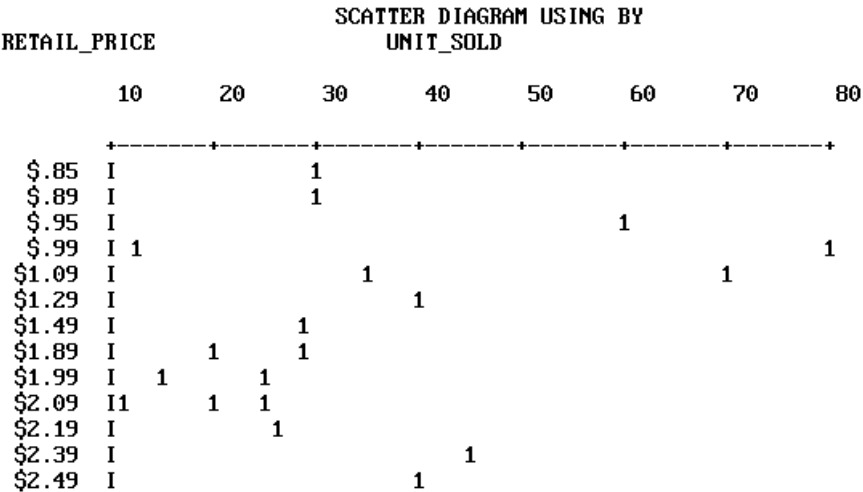
Scatter diagrams illustrate occurrence patterns and distribution of variables. You create them by issuing requests containing the verb **PRINT** and a sort phrase (**BY** or **ACROSS**). The choice of **BY** or **ACROSS** dictates the vertical or horizontal bias of the graph. The samples that follow illustrate both types.

```
GRAPH FILE SALES
HEADING CENTER
"SCATTER DIAGRAM USING ACROSS"
PRINT UNIT_SOLD ACROSS RETAIL_PRICE
END
```



When the request contains a BY phrase, the named sort control field is plotted down the vertical axis and the data values are scaled horizontally.

```
GRAPH FILE SALES
HEADING CENTER
"SCATTER DIAGRAM USING BY"
PRINT UNIT_SOLD BY RETAIL_PRICE
END
```



The vertical axis is not scaled even if the control field is numeric. Each separate value of the control field is plotted on a different line, but these are not arranged according to a numerical scale. The full range of horizontal scaling options is available (see *The Horizontal Axis: System Defaults* on page 18-40).

## Scatter Diagram Features

When multiple points fall in the same position, FOCUS displays either a number (for up to nine occurrences) or an asterisk (for more than nine occurrences).

When you specify more than one verb object (five are permitted), they are represented by the first letter of the field name. If they are not different you can assign unique symbols with AS phrases.

Scatter diagrams can display the following:

- Trend lines (available only in plots generated using ACROSS). Trend lines are calculated by Ordinary Least Squares (OLS) regression analysis and represent the line of best fit. You can add them to requests containing ACROSS phrases by setting the parameter GTREND before executing or replotting the request:

`SET GTREND=ON`

When two fields are plotted with GTREND=ON, FOCUS provides two trend lines. If more than two fields are plotted, however, FOCUS does not provide trend lines.

- Horizontal grids. You can add horizontal grid lines at the vertical class marks by setting the parameter GRID:

`SET GRID=ON`

- Vertical grids (available only in plots generated by requests using BY). You can add vertical parallel lines at the horizontal class marks of the scatter plot by setting the parameter VGRID:

`SET VGRID=ON`



# Adjusting Graph Elements

All graphs other than pie charts have horizontal and vertical axes. These axes usually have scales with adjustable upper and lower thresholds that are divided into class intervals representing quantities of data (scales are only provided when the variables named are computational fields). Class intervals are further broken down with tick marks representing smaller increments of data.

When multiple graphs are created in a single request, FOCUS determines the default horizontal scale after examining all values to be plotted, and the same scale is then applied to each graph. Vertical scales are recalculated each time, however, and adjusted for the values in each graph (unless you override this feature).

Some graph forms, notably connected point plots, histograms, and bar charts, can be visually strengthened by the addition of parallel lines across the horizontal and/or vertical axes, to form a grid against which the data is arrayed.

The material that follows describes the default conditions for all of these graph features, and the facilities for changing the default values to create customized output.

At any time during your session you can review the current GRAPH parameter settings by typing

```
? SET GRAPH
```

which displays the current settings of all of the adjustable GRAPH parameters, as shown below.

| Parameter        | Setting |
|------------------|---------|
| DEVICE           | IBM3270 |
| GTPROMPT         | OFF     |
| GRID             | OFF     |
| VGRID            | OFF     |
| HAXIS            | 130     |
| VAXIS            | 66      |
| GTREND           | OFF     |
| GRIBBON (GCOLOR) | OFF     |
| VZERO            | OFF     |
| VAUTO            | ON      |
| VMAX             | .00     |

| Parameter | Setting |
|-----------|---------|
| VMIN      | .00     |
| HAUTO     | ON      |
| HMAX      | .00     |
| HMIN      | .00     |
| AUTOTICK  | ON      |
| HTICK     | .00     |
| HCLASS    | .00     |
| VTICK     | .00     |
| VCLASS    | .00     |
| BARWIDTH  | 1       |
| BARSPACE  | 0       |
| BARNUMB   | OFF     |
| HISTOGRAM | ON      |
| HSTACK    | OFF     |
| BSTACK    | OFF     |
| PIE       | OFF     |
| GMISSING  | OFF     |
| GMISSVAL  | .00     |

**Figure 18-10. GRAPH Parameter Settings**

For information about each of the parameters listed, refer to *SET Parameters* on page 18-60.

## The Horizontal Axis: System Defaults

The width of each graph, including any surrounding text, is controlled by the HAXIS parameter. For online displays, FOCUS automatically detects the terminal width and plots the graph accordingly.

For graphs generated OFFLINE, the default value for HAXIS is normally set to the maximum possible size for the output device selected, after allowing for the inclusion of any text required for the vertical axis and its labels along the left margin. To maximize display space, you can limit the size of your labels through the use of either AS phrases or DECODE expressions.

In setting the scale (when AUTOTICK=ON, and HAUTO=ON), FOCUS determines the amount of available space and the range of values selected for plotting. It then selects minimum, intermediate, and maximum unit values for the horizontal axis scale that encompass the range of values and are convenient multiples of an appropriate power of 10 (10 vs. 1000 vs. 1,000,000).

When you select a high-resolution graphic device, FOCUS controls the axis dimensions according to the values shown for the various devices in *SET Parameters* on page 18-60.

### Syntax

#### How to Set the Width

To set the width of the graph to a given number of characters, issue the SET statement

```
SET HAXIS=nn
```

where *nn* is a numeric value between 20 and 130.

## Syntax

### How to Set the Scale: Assigning Fixed Limits

FOCUS automatically sets the horizontal scale to cover the total range of values to be plotted (HAUTO=ON). The range is divided into intervals called classes. The scale is normalized to provide class values rounded to the appropriate multiples and powers of 10 for the intervals plotted on the axis.

If you wish to assign fixed upper and lower limits (useful when producing a series of graphs where consistent scales are needed), you do so by turning off the automatic scaling mechanism and setting new limit values. This is done with the SET command. The syntax is as follows

```
SET HAUTO=OFF, HMAX=nn, HMIN=nn
```

where:

**HAUTO**

Is the automatic scaling facility.

**HMAX**

Is the parameter for setting the upper limit on the horizontal axis. The default is 0.

**HMIN**

Is the parameter that controls the lower limit on the horizontal axis when HAUTO is OFF. The default is 0.

**nn**

Is the new limit.

#### **Note:**

- When entering several SET parameters on one line, separate them with commas.
- If you define limits that do not incorporate all of the data values, FOCUS displays OVER and/or UNDER to indicate that some of the data extracted is not reflected on the graph.

## Syntax

### How to Set Class and Tick Intervals

Class intervals are the intervals between the labels and grid lines on a graph. Tick intervals are the subdivisions of class intervals. When AUTOTICK is ON, FOCUS automatically determines the class and tick intervals.

To set the class and tick intervals yourself, first turn off the default scaling mechanism; then reset the class and tick intervals with the SET command

```
SET AUTOTICK=OFF, HCLASS=nn, HTICK=nn
```

where:

**AUTOTICK**

Is the automatic scaling mechanism.

**HCLASS**

Is the parameter that controls the class interval on the horizontal axis when AUTOTICK is OFF. The default is 0.

*nn*

Is the new class interval value for the axis.

**HTICK**

Is the parameter that controls the tick interval when AUTOTICK is OFF. The default is 0.

*nn*

Is the new tick interval for the axis.

#### **Note:**

- When issuing more than one parameter with a sample SET command, separate parameters with commas as shown above.
- To make the changes apparent on the screen, SET SCREEN to PAPER.
- The number of ticks per class is HCLASS/HTICK.

## The Vertical Axis: System Defaults

The vertical axis (VAXIS) represents the number of lines in the graph, including any surrounding text.

For online displays, FOCUS automatically plots the graph according to the terminal height. For graphs generated offline, FOCUS respects VAXIS settings.

FOCUS automatically sets the vertical scale to cover the total range of values to be plotted (VAUTO=ON). The height is set as high as possible (taking into consideration the presence of any headings and/or footings, and the need to provide suitably rounded vertical class markers).

The range is divided into intervals called classes. The scale is normalized to provide class values rounded to the appropriate multiples and powers of 10 for the intervals plotted on the axis.

As with the horizontal axis, FOCUS selects the vertical axis size whenever you select a high-resolution graphic device (see *SET Parameters* on page 18-60).

### Syntax

#### How to Set the Height

Use the following SET command to set the vertical axis

```
SET VAXIS=nn
```

where *nn* is a number in the range 20-66.

### Syntax

#### How to Set the Scale: Assigning Fixed Limits

If you wish to give the vertical scale fixed upper and lower limits (useful when producing a series of graphs where consistent scales are needed), you can do so by turning off the automatic scaling mechanism and setting fixed limits. This is done with the SET command

```
SET VAUTO=OFF, VMAX=nn, VMIN=nn
```

where:

**VAUTO**

Is the automatic scaling facility.

**VMAX**

Is the parameter for setting the upper limit on the vertical axis. The default is 0.

**VMIN**

Is the parameter that controls the lower limit on the vertical axis when VAUTO is OFF. The default is 0.

**nn**

Is the new limit.

**Note:**

- When entering several SET parameters on one line, separate them with commas.
- If you define limits that do not incorporate all of the data values, FOCUS displays OVER and/or UNDER to indicate that some of the data extracted is not reflected on the graph.

## Syntax

### How to Set Class and Tick Intervals

To set the class and tick intervals on the vertical axis, first turn off the default scaling mechanism, and then reset the class and tick intervals with the SET command

```
SET AUTOTICK=OFF, VCLASS=nn, VTICK=nn
```

where:

**AUTOTICK**

Is the automatic scaling mechanism.

**VCLASS**

Is the parameter that controls the class interval on the vertical axis when AUTOTICK is OFF. The default is 0.

*nn*

Is the new class interval for the vertical axis.

**VTICK**

Is the parameter that controls the tick interval when AUTOTICK is OFF. The default is 0.

*nn*

Is the new tick interval for the axis.

**Note:**

- When setting more than one parameter, separate them with commas.
- To make the changes apparent on screen, SET SCREEN to PAPER.
- The number of ticks per class is VCLASS/VTICK.

## Highlighting Facilities

FOCUS contains several facilities for highlighting the information shown on your graphs. Specifically, these include the following:

- Grid lines can be added on one or both axes of connected point plots and scatter diagrams or the horizontal axis of histograms.
- Trend lines are usually included on most scatter plots.
- Summary numbers can be printed for each slice of a pie chart or bar on a bar chart.

### Syntax

#### How to Add Horizontal or Vertical Grids

Grids often make graphs easier to read. They are parallel lines drawn across the graph at the vertical and/or horizontal class marks on the axes.

Horizontal grid lines are available on connected point plots, histograms, and scatter diagrams. To add them at the vertical class marks on your graph, issue the following command:

```
SET GRID=ON
```

Vertical grid lines are available only on high-resolution devices in requests for connected point plots and scatter diagrams, and only when the values on both axes are numeric. To add them at the horizontal class marks on the graph, issue the following command:

```
SET VGRID=ON
```

To remove the lines, set the appropriate parameter OFF.

### Syntax

#### How to Add Summary Numbers in Pie and Bar Charts

To print a summary number at the end of each bar on a bar chart or in each slice of a pie chart, set the parameter BARNUMB

```
SET BARNUMB=ON
```

These summary numbers are not available on histograms.



## Syntax

### How to Add Trend Lines on Scatter Plots

Trend lines are useful on scatter plots to give a focus to the sometimes confusing array of plot points. The trend line represents the notion of the “best fit” calculated by Ordinary Least Squares (OLS) regression analysis.

When two data fields are scattered across the same horizontal axis, each is given its own trend line. On some terminals with two-color ribbons the lines are differentiated by color.

The system always requests a value for the parameter GTREND, whenever a scatter diagram is requested (the default value for GTREND is OFF). To request a trend line set GTREND on:

```
SET GTREND=ON
```

## Special Topics

There are a few topics that have general applicability for many graph applications. Specifically, they are:

- How does FOCUS handle dates in graphs?
- How is missing data handled?
- Is it possible to save formatted graphic output and display it later?
- Is it possible to send graphs to a Personal Computer for display?
- What is the nature of the interface between FOCUS and CA-TELLAGRAF?
- What is the nature of the interface between FOCUS and ICU (Interactive Chart Utility)?

These are described in the following sections.

## Plotting Dates

Numerical fields containing dates are recognized by FOCUS through the formats in their Master Files. Such fields are interpreted by FOCUS if you name them in ACROSS or BY phrases in GRAPH requests. To review the various format types, see the *Describing Data* manual.

When plotting dates, FOCUS handles them in the following manner:

- If the date field named has a month format, it is plotted in ascending time order (even though the file is not sorted in ascending date order). Hence, month/year values of 01/76, 03/76, 09/75 will be plotted by month within year: 09/75, 01/76, 03/76.
- Axis scaling is performed on the basis of days in the month and months in the year. When the date format includes the day, the scale usually starts at the first day of the month as the zero axis point.

In some instances you may wish to selectively combine groups of date point plots to reduce the number of separate points on the horizontal axis. You do this with the IN-GROUPS-OF option. For example, if the date field format is I6YMD you can display the data by month rather than by day by grouping it in 30-day increments:

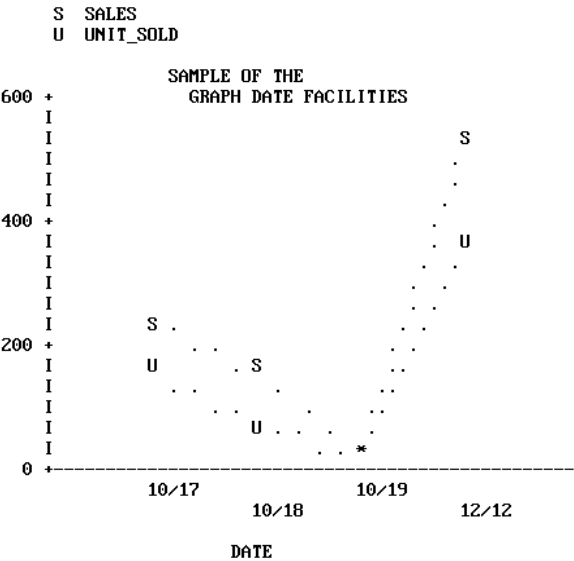
```
ACROSS DATE IN-GROUPS-OF 30
```

This eliminates plot points for individual days. If your date format is in a legacy YMD format you could also redefine the format and divide the field contents by 100 to eliminate the days:

```
DATE/I4YM=DATE/100
```

The example that follows illustrates a graph with date plots.

```
GRAPH FILE SALES
HEADING
"</6 <22 SAMPLE OF THE"
"<24 GRAPH DATE FACILITIES"
SUM SALES AND UNIT_SOLD ACROSS DATE
END
```



## Handling Missing Data

You can handle missing data selectively in GRAPH requests. You can portray the missing data as null values or you can choose to ignore missing values and have the plot span the missing points. This applies to requests containing both ACROSS and BY phrases, where the ACROSS values are plotted across the horizontal axis.

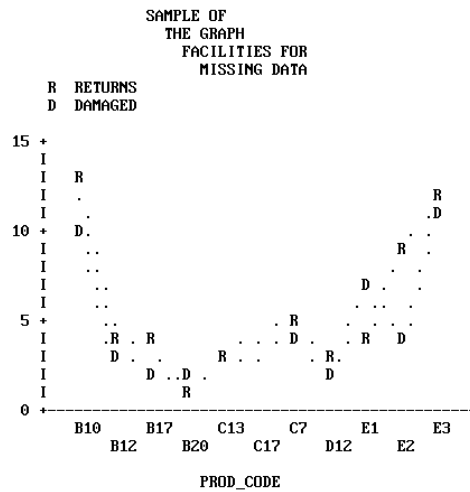
Normally, missing values on the vertical axis are ignored (VZERO=OFF). If ON, the values are treated as zero (0).

You instruct the system to ignore missing values through the SET options, GMISSING and GMISSVAL, or you can set GPROMPT=ON, and select the processing of the missing values when you execute your request. These SET operations can be done once for your entire session, or may be done on an individual basis to refine a particular request. Keep in mind that they remain in effect until you reset the parameters (see *SET Parameters* on page 18-60).

The examples that follow illustrate the same request, but with different treatments of missing values selected.

```
SET GMISSING=ON, HIST=OFF
```

```
GRAPH FILE SALES
HEADING
"</1 <22 SAMPLE OF"
"<24 THE GRAPH "
"<26 FACILITIES FOR"
"<28 MISSING DATA"
SUM RETURNS AND DAMAGED ACROSS PROD_CODE
END
```

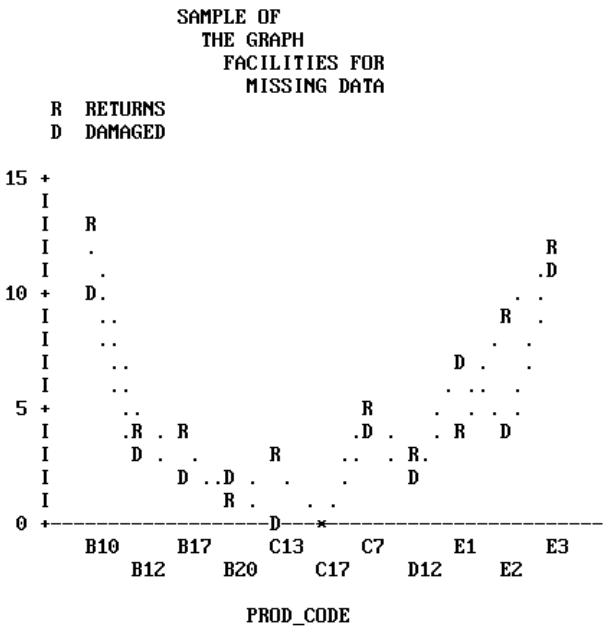


In this example GMISSING is ON and GMISSVAL is 0, so the graph ignores zero values for products C13 and C17.

The graph below shows the effect of changing the GMISSING parameter to OFF.

```
SET GMISSING=OFF
```

```
REPLOTT
```



The values for products C13 and C17 were shown as positive values with GMISSING ON. With GMISSING=OFF, the zero values for products C13 and C17 are plotted on the graph.

## Using Fixed-Axis Scales

When creating series of graphs it is often desirable to have the same horizontal and vertical scales used for each graph in the group. This situation arises whenever your graph request combines an ACROSS phrase with a BY phrase.

In such requests the ACROSS values are plotted across the horizontal axis and a separate graph is created for each value of the BY field. The default scales for the graphs will vary depending on the range of values for each verb object and BY field combination.

To apply the same scale to each graph, turn off the default scaling mechanisms, and define your own minimum and maximum values for the axis thresholds (see *How to Set the Scale: Assigning Fixed Limits* on page 18-41 and *How to Set the Scale: Assigning Fixed Limits* on page 18-43).

## Saving Formatted GRAPH Output

You can place the output from GRAPH commands into specially formatted SAVE files for subsequent conversion into printed or displayed graphs. This capability (called deferred output) is useful for developing graph requests on a device other than the one you will use to produce the final graph.

The facility described below is available for all ASCII graphics devices that FOCUS supports, but is not available for the IBM 3279 color graphics terminal, which has a separate GDDM facility for this purpose (see *IBM Devices Using GDDM* on page 18-55). In addition, deferred output cannot be generated from a CONSOLE.

The syntax for the FOCUS facility is

```
GRAPH FILE ...
SUM ...
.
.
.
ON {GRAPH|TABLE} SAVE [AS savename] FORMAT GRAPH
ON {GRAPH|TABLE} SET parameter value [, parameter value...]
END
```

where:

**ON GRAPH|ON TABLE**

Denotes the command environment from which the request is entered. This syntax suppresses display of the output and returns a message that the file has been saved.

**SAVE|SET**

Is the action taken.

**AS savename**

Is an optional parameter that allows you to assign a permanent file name as the target for the formatted output. The default is FOCSAVE.

**parameter value**

Is the system value you want to change or set. Any parameter discussed in the *Developing Applications* manual can be set or changed here. The syntax is essentially the same as ON TABLE SET, which is discussed in Chapter 4, *Sorting Tabular Reports*.

**FORMAT GRAPH**

Specifies that the output is to be formatted for whatever graphics device is specified in the DEVICE parameter (see *SET Parameters* on page 18-60), and saved in either the SAVE file or a file you name in an AS phrase.

As an alternative, you can display a graph on the CONSOLE before creating a specially formatted SAVE file. To use this facility, enter a GRAPH request to generate a display, as shown below:

```
GRAPH FILE ...
SUM ...
.
.
.
END
```

After viewing the graph, use the following syntax to save the graph for later output on another device:

```
SAVE [AS savename] FORMAT GRAPH
```

## Syntax

## How to Display Stored Graphs

To display stored graphs, issue the appropriate form of the REPLOT command from the output graphics DEVICE

```
REPLOT [GRAPH|FROM] ddname
```

where:

```
REPLOT [GRAPH|FROM]
```

Is the function to be performed.

```
ddname
```

Is the SAVE file name. This must be provided even if the default FOCSAVE file was used.

### Note:

- You need not redefine the graphics device with another SET command. The device specified through the DEVICE= parameter when the graph was saved still applies.
- You can save the internal matrix produced for a request and issue a REPLOT later in the session if SAVEMATRIX is set to ON (see the *Developing Applications* manual).

You can allocate the file yourself through the appropriate operating system procedure or you can let FOCUS allocate the SAVE file for you dynamically. If you allow FOCUS to allocate the file it will allocate a temporary file that you must rename if you wish to keep it after you log off.

The record layout of the graphics SAVE file is documented in Technical Memorandum #7704 *Description of Deferred Graphics Output* (available through your Information Builders Branch Office). You can process this file yourself if you have a deferred graph system that accepts low-level terminal graphics commands.

## Displaying Graphs With PC/FOCUS or FOCUS for Windows

If you have FOCUS on the mainframe and PC/FOCUS® on your personal computer and an appropriate communications link, you can extract data from the mainframe files and send the extract file to the personal computer where you actually issue the GRAPH request in PC/FOCUS or FOCUS for Windows. This is a useful option because the printers attached to personal computers often create more attractive graphs than those produced on the system high-speed printer. FOCUS file transfer facilities are described in the *Developing Applications* manual.

Note, however, that you cannot send formatted graphs to the personal computer for plotting.

## Creating Formatted Input for CA-TELLAGRAF

The Interface to CA-TELLAGRAF is a separate optional interface product that you use to create formatted FOCUS output files ready for processing by CA-TELLAGRAF, the publication-quality graphics system produced by Computer Associates.

With it, you can write FOCUS GRAPH requests that generate files containing actual CA-TELLAGRAF commands and all of the necessary data and control information for producing graphs.

The data may originate in any FOCUS file or any file that FOCUS can read (for example, QSAM, VSAM, ISAM, IMS, CA-IDMS/DB, ADABAS, TOTAL, SQL, SYSTEM 2000, Model 204).

Directions for using the Interface can be found in the *TELLAGRAF Interface Users Manual*.



## Using the FOCUS ICU Interface

The FOCUS ICU Interface is a separate optional interface product that you can use to generate graphs in conjunction with IBM's Interactive Chart Utility.

ICU displays graphs and provides menu selections which allow you to change such factors as graph type, size, and legend, and to send the graph to a printer.

The ICU Interface can place you directly in the ICU environment or can save the graph format and data for subsequent ICU processing. When you leave ICU, control is returned to FOCUS.

All ICU graphics requests follow the standard FOCUS rules and each of the default graphs is represented by an ICU format file distributed with FOCUS.

To use the ICU Interface, issue the command:

```
SET DEVICE = ICU
```

Subsequent GRAPH requests will use ICU to generate graphs.

Directions for using this Interface can be found in the *ICU Interface Users Manual*.

## Special Graphics Devices

Graphs created with the FOCUS graphics generator can be printed or displayed in three levels of detail:

- Low-resolution graphs are produced by high-speed line printers and non-graphics terminals. Normally, this is adequate graphic information. While such graphs are not elegant, they are easily produced and allow you to preview graphic scenarios and refine the shapes and contents of your graphs. Subsequently, to create more "finished" versions you need only choose a different device or save the formatted output in a file to print later when a high-resolution device is available.
- Medium-resolution graphs are produced on devices such as Diablo, Trendata, and Anderson-Jacobson printers. These devices, which are driven by step motors, draw nearly continuous line plots, but the quality is not adequate for presentations.
- High-resolution graphic devices print continuous line plots, smooth curves, and create presentation-quality graphs. This category includes both devices created specifically for displaying graphics images (flat-bed and continuous line plotters, and color printers), as well as color CRTs. FOCUS supports three types of high-resolution graphics devices:
  - Hewlett-Packard four- and eight-pen plotters.
  - IBM graphic CRTs and printers.
  - Tektronix CRTs.

An additional option, if you have FOCUS on your PC, is to create an extract file with FOCUS and send it to your PC where you create the actual graph.

## Medium-Resolution Devices

These devices use step motors to drive platens back and forth across the pages to draw two series of spaced dots that simulate continuous lines. There are separate device symbols for the most frequently used printers (see `DEVICE` in *SET Parameters* on page 18-60), and a generic device code, `HIGHRES` (or `HIGHRS12` for 12 pitch), for use with many unlisted printers.

Pie charts are not available on these devices.

When using this type of printer, set `PAUSE=ON` so that you can adjust the paper in the printer before drawing the graph.

## High-Resolution Devices

This section describes the special considerations that apply when directing your FOCUS graphs to high-resolution devices from IBM, Hewlett-Packard, and Tektronix.

### IBM Devices Using GDDM

To produce graphs on IBM graphics printers or high-resolution graphics terminals you must have IBM's Graphical Data Display Manager (GDDM). GDDM provides various subroutines for saving, printing, and copying graphic screen contents. FOCUS produces graphs on IBM terminals or printers when you set `DEVICE=IBM3279`. See your IBM representative concerning the proper configuration for your device controller and terminal.

### GDDM Default Conditions

Whenever graphs are created using FOCUS and GDDM, the printed form of the graph (activated by pressing the PF4 key) has a default size of 132 by 80 characters on 3284 or 3287 printers. These sizes are independent of the parameters that control the lengths of the axes. As a default, each graph is presented with a frame (border). If you wish to omit the frame, set `FRAME=OFF`.

### GDDM Save and Print Facilities

GDDM includes facilities for saving generated graphs that you activate by pressing the PF1 key to save graphs in an `ADMSAVE` file on your operating system. Thus saved, you can subsequently use the IBM program `ADMUSF2` (supplied with GDDM) to display the saved screens.

For special instructions covering the positioning of graphs on IBM 3284 or 3287 printers, please refer to Technical Memorandum #7689, *Plot Table Settings* (available through your Information Builders Branch Office).

## Graphics Device Characteristics

To draw vectors, use 7-color displays, or define your own special field patterns, you need a 3279 Model 2B, 3B, 3SG, or 3X with a 3274 terminal controller and C configuration support. C supports structured field and attribute processing (SFAP) and the use of programmed symbols (PS). (The Model 3276 terminal controller does not use C.)

3279 Models 2A and 3A have only Base Color which automatically maps colors to preset 3270 field types:

- Protected intensified becomes white.
- Unprotected intensified becomes red.
- Protected normal intensity fields become blue.
- Unprotected normal intensity fields become green.

Thus, FIDEL is automatically color coded with no programming changes, but only in a base color. Additional colors are available with the 3SG, 3X and the older B models.

## Hewlett-Packard Plotters

The Hewlett-Packard 7220 series plotters translate FOCUS graph requests into 4- or 8-color graphs, suitable for presentations. Color selections and assignments are made using the standard Hewlett-Packard procedures. (Special pens are available from Hewlett-Packard for plotting graphs on transparencies for overhead projection.) For plotters with optional text facilities FOCUS has special parameters for controlling:

- Text positioning (column, line, and spacing).
- Color pen selection (red, blue, green, black).
- Letter sizes (two or four times the default size).
- Special font selection (slanted text).

To activate Hewlett-Packard plotters use the appropriate form of the SET TERM or DEVICE (see *SET Parameters* on page 18-60). FOCUS provides default axes lengths and scaling, but these and the other graphic elements can all be changed by adjusting SET parameters discussed in *Adjusting Graph Elements* on page 18-38 and summarized in *SET Parameters* on page 18-60.

Ordinarily, plotters are connected in line with a terminal and a modem. Thus, you can refine your graph requests, viewing the output on the terminal, until you produce exactly what you want and then set the DEVICE parameter to your plotter and issue the REPLOT command to produce the hard copy.

Use the plotter controls to position graphs anywhere on sheets of paper up to 11 by 16.5 inches. Unless you change the default paper size, FOCUS prepares output for an 8.5 by 11 inch sheet placed lengthwise in the lower left-hand corner of the plotter. The other default assignments are as follows:

`HAXIS=130, VAXIS=66, GCOLOR=ON`

## Tektronix Color Terminals

Tektronix high-resolution CRTs can display the output from GRAPH requests, but only in black and white. The sizes of the vertical and horizontal axes are set depending on the device selected and cannot be overridden. Select the appropriate device number from those listed in *SET Parameters* on page 18-60.

# Command and SET Parameter Summary

The FOCUS GRAPH command plots data retrieved with request statements in the form of a graph, with horizontal and vertical axes. Many of the elements used in TABLE requests are used in exactly the same way in GRAPH requests.

The GRAPH environment also includes a set of parameters that control the look of the graph and offer some additional control at run time (for example, pause to adjust paper before printing, select a device, etc.).

## GRAPH Command

In the syntax samples that follow, the elements are the same as those used in TABLE requests. The complete set is shown here but the elements are described more fully in Chapter 4, *Sorting Tabular Reports*.

### Syntax

#### How to Enter the Environment

To enter the GRAPH environment enter the following:

```
GRAPH FILE filename
```

### Syntax

#### How to Specify Annotating Text

Heading strings can contain any character except the double quotation mark (“), and can also contain field references and formatting controls.

Heading—This syntax is used to specify graph headings:

```
[HEADING [CENTER]]
"string1"
["string2"]
```

Field reference format—This syntax is used to specify field reference format:

```
<[prefix.]fieldname[>]
```

Formatting Controls—The following formatting controls may be specified as part of a graph request:

```
Tab to column "n"<n
Tab "n" columns to the right<+n
Tab "n" columns to the left<-n
Return to column 1
 and advance "n" lines.</n
Name a color for a line<.color
Select special font
 [BIG, SLANT or BLOCK on HP7220]. .<.fontname
 ("BIG" doubles the character
 sizes, "BLOCK" quadruples them)
Reset controls to default settings <.CLEar
```

## Syntax

### How to Name the Subject and Graph Type

The following syntax is used to specify the subject and graph type:

```
{command} object1 [[AND|OVER] object2...object5]
```

where:

*command*

Is one of the following: PRINT, WRITE, SUM, ADD or COUNT.

## Syntax

### How to Specify Display Fields

Display fields can be any of the following:

```
[prefix.]fieldname [AS 'string'] [IN position]
COMPUTE name1 [/format1] = expression1:[AS 'string1']
COMPUTE name2 [/format2] = expression2:[AS 'string2']
```

## Syntax

### How to Specify Horizontal Sorting of Data Points

The following syntax is used for horizontal sorting of data:

```
ACROSS fieldname [IN-GROUPS-OF n [TOP]][AS 'string']
ACROSS fieldname [IN position]
```

## Syntax      How to Specify Separate Graphs or Vertical Sorting of Plot Points

The following syntax is used for specifying separate graphs or vertical sorting of plot points:

```
BY fieldname [IN-GROUPS-OF n [TOP]][AS 'string']
```

## Syntax      How to Save the Formatted Graph Data in a File

The following syntax is used to save formatted graph data in a file:

```
ON [GRAPH] SAVE [AS filename] FORMAT GRAPH
```

## Syntax      How to Complete the GRAPH Request

To complete a graph request, type the command END on a separate line:

```
END
```

If you do not wish to complete the graph request, use one of the following methods to abort the request and return to FOCUS:

- To quit in the middle of a graph request, type the command QUIT on a separate line:

```
QUIT
```

- To terminate the display of a graph, type the command HT from the command line:

```
HT
```

## Syntax      How to Concatenate Unlike Data Sources

To concatenate unlike data sources in a single graph request, divide your request into one main request that retrieves the first file and a subrequest for each concatenated file. The main request defines the data fields, sorting criteria, and output format for each file. The MORE command concatenates each file after the first. The syntax is:

```
GRAPH FILE file1
 main request
MORE
FILE file2
 subrequest
MORE
.
.
.
END
```

**Note:** IF and WHERE selection tests apply only to the subrequest in which they appear.

## SET Parameters

To set the parameters that control the GRAPH environment, use the appropriate variation of the SET command. The syntax is as follows:

```
SET parameter=value,parameter=value...
```

For example:

```
SET HAXIS=75,VAXIS=40
SET GRID=OFF,BARSPACE=2,BARWIDTH=3
```

**Note:**

- Repeat the command SET on each new line.
- When entering more than one parameter on a line, separate them with commas.
- You can use unique truncations of parameter names. You must make sure that they are unique.

To review the current parameter settings, issue the command

```
? SET GRAPH
```

which produces a listing of the values.

The table that follows lists all of the parameters in alphabetic sequence, showing the name, range of values (default is underlined), and function of each.

| Parameter Name | Range of Values | Parameter Function                                                                        |
|----------------|-----------------|-------------------------------------------------------------------------------------------|
| AUTOTICK       | <u>ON</u> /OFF  | When ON, FOCUS automatically sets the tick mark intervals. (See also HTICK and VTICK.)    |
| BARNUMB        | ON/ <u>OFF</u>  | Places the summary values at the ends of the bars on bar charts, or slices on pie charts. |
| BARSPACE       | 0- <u>20</u>    | Specifies the number of lines separating the bars on bar charts.                          |
| BARWIDTH       | 1- <u>20</u>    | Specifies the number of lines per bar on bar charts.                                      |
| BSTACK         | ON/ <u>OFF</u>  | Specifies that the bars on a bar chart are to be stacked rather than placed side by side. |

| Parameter Name                    | Range of Values          | Parameter Function                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DEVICE or<br>TERMINAL             | <a href="#">IBM3270</a>  | Specifies the plotting device or terminal to be used. When the default is used, low-resolution graphics are sent to your terminal or to the printer if PRINT=OFFLINE (see the SET command in the <i>Developing Applications</i> manual). Medium- and high-resolution devices are selected by entering one of the following parameter settings for the DEVICE (or TERMINAL). |
| <b>Medium-resolution devices:</b> |                          |                                                                                                                                                                                                                                                                                                                                                                             |
|                                   | <a href="#">AJ</a>       | Specifies Anderson Jacobson - Model AJ830.                                                                                                                                                                                                                                                                                                                                  |
|                                   | <a href="#">AJ12</a>     | Specifies Anderson Jacobson - Model AJ832 (12 Pitch).                                                                                                                                                                                                                                                                                                                       |
|                                   | <a href="#">DIABLO</a>   | Specifies Diablo - Model 1620.                                                                                                                                                                                                                                                                                                                                              |
|                                   | <a href="#">DIABLO12</a> | Specifies Diablo - Model 1620 (12 Pitch).                                                                                                                                                                                                                                                                                                                                   |
|                                   | <a href="#">GS</a>       | Specifies Gencom.                                                                                                                                                                                                                                                                                                                                                           |
|                                   | <a href="#">GS12</a>     | Specifies Gencom (12 Pitch).                                                                                                                                                                                                                                                                                                                                                |
|                                   | <a href="#">HIGHRES</a>  | Specifies generic device for most medium-resolution graphic devices.                                                                                                                                                                                                                                                                                                        |
|                                   | <a href="#">HIGHRS12</a> | Specifies generic device -see above (12 Pitch).                                                                                                                                                                                                                                                                                                                             |
|                                   | <a href="#">TRENDATA</a> | Specifies Trendata - Model 4000A.                                                                                                                                                                                                                                                                                                                                           |
|                                   | <a href="#">TRENDT12</a> | Specifies Trendata - Model 4000A (12 Pitch).                                                                                                                                                                                                                                                                                                                                |



| Parameter Name                                       | Range of Values | Parameter Function                                                                                                                       |
|------------------------------------------------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>High-resolution devices from Hewlett-Packard:</b> |                 |                                                                                                                                          |
|                                                      | HP7220          | Specifies HP Models 7229A and 7470A. Both are 4-pen plotters with no paper advance. Model 7470 requires a special Y cable (Part #17455). |
|                                                      | HP7220S         | Specifies HP Model 7220S, 4-pen plotter with paper advance.                                                                              |
|                                                      | HP7220C         | Specifies HP Models 7220C and 7475A. Both are 8-pen plotters with no paper advance. Model 7475 requires a special Y cable (Part #17455). |
|                                                      | HP7220T         | Specifies HP Model 7220T, 8-pen plotter with paper advance.                                                                              |
|                                                      | HP7221          | Specifies HP Model 7221, 4-pen plotter with no paper advance.                                                                            |
|                                                      | HP7221S         | Specifies HP Model 7221S, 4-pen plotter with paper advance.                                                                              |
|                                                      | HP7221C         | Specifies HP Model 7221C, 8-pen plotter with no paper advance.                                                                           |
|                                                      | HP7221T         | Specifies HP Model 7221T, 8-pen plotter with paper advance.                                                                              |

**Note:** The default horizontal and vertical axes for all Hewlett-Packard devices are as follows:

HP7220, HP7220S, HP7220C, HP7220T, HP7221, HP7221S, HP7221C, HP7221T

| Parameter Name                           | Range of Values | Parameter Function                                                                                                                                                                                                                                                                                                                |
|------------------------------------------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>High-resolution devices from IBM:</b> |                 |                                                                                                                                                                                                                                                                                                                                   |
|                                          | IBM3279         | Specifies one of the following devices:<br><br>Any IBM 3270 series device that supports GDDM graphics, such as the 3279-S3G, 3179, or 3472. This includes PCs with fully compatible 3270 series hardware and software.<br><br>Printers: Any IBM 3270 series printer that supports GDDM graphics such as the 3287-2C and the 4224. |

**Note:** IBM's Graphical Data Display Manager (GDDM) is required for all of these devices. Entering this value automatically sets the following GRAPH parameter values: HAXIS=80, VAXIS=32, and GCOLOR=ON. For any monochrome device, you should set GCOLOR=OFF; for any Model 2 device (24x80 screen), you should set VAXIS=-24.

| Parameter Name                                 | Range of Values | Parameter Function                                                                                                                          |
|------------------------------------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <b>High-resolution devices from Tektronix:</b> |                 |                                                                                                                                             |
|                                                | TEK4010         | Specifies one of the following models: 4010, 4050 series and 4100 series (B/W only). Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF. |
|                                                | TEK4012         | Specifies Model 4012. Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF.                                                                |
|                                                | TEK4013         | Specifies Model 4013. Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF.                                                                |
|                                                | TEK4014         | Specifies Model 4014. Automatically sets HAXIS=133, VAXIS=64, and GCOLOR=OFF.                                                               |
|                                                | TEK4014E        | Specifies Model 4014E. Automatically sets HAXIS=133, VAXIS=64, and GCOLOR=OFF.                                                              |
|                                                | TEK4015         | Specifies Model 4015. Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF.                                                                |

| Parameter Name                                 | Range of Values | Parameter Function                                                                                                                                                                                                                              |
|------------------------------------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>High-resolution devices from Tektronix:</b> |                 |                                                                                                                                                                                                                                                 |
|                                                | TEK4015E        | Specifies Model 4015E. Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF.                                                                                                                                                                   |
|                                                | TEK4025         | Specifies Model 4025. Automatically sets HAXIS=80, VAXIS=32, and GCOLOR=OFF.                                                                                                                                                                    |
|                                                | TEK4027         | Specifies Model 4027. Automatically sets HAXIS=80, VAXIS=32, and GCOLOR=ON.                                                                                                                                                                     |
|                                                | TEK4662         | Specifies Model 4662. Plot address is D. It is recommended that you set GCOLOR=OFF, HAXIS=80, VAXIS=32. If the Model 4662 is connected to a Model 4025, set DEVICE=TEK4025. If the Model 4662 is connected to a Model 4027, set DEVICE=TEK4027. |

| Parameter Name         | Range of Values | Parameter Function                                                                                                                                                                                                                                                                                                         |
|------------------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FRAME                  | ON/OFF          | For GDDM graphics, ON (the default) indicates you want a frame around your graph. To omit the Frame, set OFF.                                                                                                                                                                                                              |
| GCOLOR<br>(or GRIBBON) | ON/OFF          | On medium- and high-resolution devices, setting this parameter OFF causes different black and white patterns to be substituted for colors. On medium-resolution devices, setting it ON causes alternation between black and red ribbons on multiline plots.<br><b>Note:</b> 3287 printers use black, red, blue, and green. |
| GMISSING               | ON/OFF          | If ON, specifies that variables with the value specified in GMISSVAL are to be ignored.                                                                                                                                                                                                                                    |

| Parameter Name | Range of Values | Parameter Function                                                                                                                                                                                                                                 |
|----------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GMISSVAL       | <i>nn</i>       | Specifies the variable value that represents missing data.                                                                                                                                                                                         |
| GPROMPT        | ON/ <u>OFF</u>  | When ON, FOCUS prompts for all pertinent graph parameters.                                                                                                                                                                                         |
| GRIBBON        |                 | See GCOLOR.                                                                                                                                                                                                                                        |
| GRID           | ON/ <u>OFF</u>  | When ON, specifies that a grid of parallel horizontal lines is to be drawn on the graph at the vertical class marks (see also VGRID).                                                                                                              |
| GTREND         | ON/ <u>OFF</u>  | When ON, specifies that a trend line is to appear on scatter diagrams.                                                                                                                                                                             |
| HAUTO          | <u>ON</u> /OFF  | Specifies automatic scaling of the horizontal axis unless overridden by the user. If OFF, user must supply values for HMAX and HMIN.                                                                                                               |
| HAXIS          | 20- <u>130</u>  | Specifies the width in characters of the horizontal axis. This parameter can be adjusted for graphs generated OFFLINE. HAXIS is ignored for ONLINE displays since FOCUS automatically adjusts the width of the graph to the width of the terminal. |
| HCLASS         | <i>nnn</i>      | Specifies the horizontal class interval when AUTOTICK=OFF.                                                                                                                                                                                         |
| HISTOGRAM      | ON/ <u>OFF</u>  | When ON, FOCUS draws a histogram instead of a curve when values on the horizontal axis are not numeric.                                                                                                                                            |
| HMAX           | <i>nnn</i>      | Specifies the maximum value on the horizontal axis when the automatic scaling is not used (HAUTO=OFF).                                                                                                                                             |
| HMIN           | <i>nnn</i>      | Specifies the minimum value on the horizontal axis when the automatic scaling is not used (HAUTO=OFF).                                                                                                                                             |
| HSTACK         | ON/ <u>OFF</u>  | Specifies that the bars on a histogram are to be stacked rather than placed side by side.                                                                                                                                                          |

| Parameter Name | Range of Values        | Parameter Function                                                                                                                                                                                                                                                                                                                                               |
|----------------|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HTICK          | <i>nnn</i>             | Specifies the horizontal axis tick mark interval, when AUTOTICK is OFF.                                                                                                                                                                                                                                                                                          |
| PAUSE          | ON/ <u>OFF</u>         | Specifies whether there will be a pause for paper adjustment on the plotter after the request is executed.                                                                                                                                                                                                                                                       |
| PIE            | ON/ <u>OFF</u>         | Specifies a pie chart is desired (only available on high-resolution devices).                                                                                                                                                                                                                                                                                    |
| PLOT           |                        | <p>Specifies the width and height settings for a graphic printer if DEVICE=IBM3279 or HIGHRES. Hexadecimal values must be supplied. For example</p> <p><i>SET PLOT=0050,0018</i></p> <p>produces a printed plot 80 by 24 decimal characters (50 hex = 80 decimal, 18 hex = 24 decimal).</p> <p>When used, the PLOT parameter must be the last parameter set.</p> |
| PRINT          | <u>ONLINE</u> /OFFLINE | When OFFLINE is entered, the graph is printed on the system high-speed printer.                                                                                                                                                                                                                                                                                  |
| TERM           |                        | See DEVICE.                                                                                                                                                                                                                                                                                                                                                      |
| VAUTO          | <u>ON</u> /OFF         | Specifies automatic scaling of the vertical axis unless overridden by the user. If OFF, user must supply values for VMAX and VMIN.                                                                                                                                                                                                                               |
| VAXIS          | 20- <u>66</u>          | Page length in lines. This parameter can be adjusted for graphs generated OFFLINE. VAXIS is ignored for ONLINE displays since FOCUS automatically adjusts the height of the graph to the height of the terminal.                                                                                                                                                 |
| VCLASS         | <i>nnn</i>             | Specifies the vertical class interval when AUTOTICK=OFF.                                                                                                                                                                                                                                                                                                         |
| VGRID          | ON/ <u>OFF</u>         | When ON, specifies that a grid of parallel vertical lines is to be drawn on the graph at the horizontal class marks (see also GRID).                                                                                                                                                                                                                             |

| Parameter Name | Range of Values | Parameter Function                                                                                               |
|----------------|-----------------|------------------------------------------------------------------------------------------------------------------|
| VMAX           | <i>nnn</i>      | Specifies the maximum value on the vertical axis when the automatic scaling is not used (VAUTO=OFF).             |
| VMIN           | <i>nnn</i>      | Specifies the minimum value on the vertical axis when the automatic scaling is not used (VAUTO=OFF).             |
| VTICK          | <i>nnn</i>      | Specifies the vertical axis tick mark interval, when AUTOTICK is OFF.                                            |
| VZERO          | ON/ <u>OFF</u>  | Normally missing values on the vertical axis are ignored (VZERO=OFF). If ON, the values are treated as zero (0). |

---

## CHAPTER 19

# Using SQL to Create Reports

### Topics:

- Supported and Unsupported SQL Statements
- Using SQL Translator Commands
- SQL Translator Support for Date, Time, and Timestamp Fields
- Index Optimized Retrieval
- TABLEF Optimization
- SQL INSERT, UPDATE, and DELETE Commands

SQL users can issue report requests that combine SQL statements with TABLE formatting phrases to take advantage of a wide range of report preparation options.

These combined requests are supported through the SQL Translator, which converts ANSI Level 2 SQL statements into executable FOCUS requests.

You can use the SQL Translator to retrieve and analyze FOCUS and DBMS data.

## Supported and Unsupported SQL Statements

SQL Translation Services is ANSI Level 2 compliant. This facility supports many, but not all, SQL statements. iWay and specific RDBMS engines may also support the *alpha1 CONCAT alpha2'* syntax. For details, see *Supported SQL Statements* on page 19-2 and *Unsupported SQL Statements* on page 19-4.

Many of the supported SQL statements are requests that are candidates for Dialect Translation. Dialect Translation is a feature that enables a server to route inbound SQL requests to SQL-capable sub-servers and data adapters wherever possible. Dialect Translation avoids translation to iWay Data Manipulation Language (DML), while maintaining data location transparency. Dialect Translation transforms a standard SQL statement into one that can be processed by the destination SQL engine, while preserving the semantic meaning of the statement.

**Note:** Because the SQL Translator is ANSI Level 2 compliant, some requests that worked in prior releases may no longer work.

### Reference

### Supported SQL Statements

SQL Translation Services supports the following:

- SELECT including SELECT ALL and SELECT DISTINCT.
- CREATE TABLE. The following data types are supported for CREATE TABLE: REAL, DOUBLE PRECISION, FLOAT, INTEGER, DECIMAL, CHARACTER, SMALLINT, DATE, TIME, and TIMESTAMP.
- INSERT, UPDATE, and DELETE for relational, IMS, and FOCUS data sources.
- Equijoins and non-equijoins.
- Outer JOINS, subject to certain restrictions. See *SQL Joins* on page 19-8 for details.
- CREATE VIEW and DROP VIEW.
- PREPARE and EXECUTE.
- Delimited identifiers of table names and column names. Table and column names containing embedded blanks or other special characters in the SELECT list should be enclosed in double quotation marks.
- Column names qualified by table names or by table tags.
- The UNION [ALL], INTERSECT [ALL], and EXCEPT [ALL] operators.
- Non-correlated subqueries for all requests in the WHERE predicate and in the FROM list.
- Correlated subqueries for requests that are candidates for Dialect Translation to an RDBMS that supports this feature. Note correlated subqueries are not supported for FOCUS and other non-relational data sources.



- Numeric constants, literals, and expressions in the SELECT list.
- Scalar functions for queries that are candidates for Dialect Translation if the RDBMS engine supports the scalar function type. These include: ABS, CHAR, CHAR\_LENGTH, CONCAT, COUNTBY, DATE, DAY, DAYS, DECIMAL, EDIT, EXTRACT, FLOAT, HOUR, IF, INT, INTEGER, LCASE, LENGTH, LOG, LTRIM, MICROSECOND, MILLISECOND, MINUTE, MONTH, POSITION, RTRIM, SECOND, SQRT, SUBSTR (or SUBSTRING), TIME, TIMESTAMP, TRIM, VALUE, UCASE, and YEAR.

Note that the following functions are not supported by FOCUS for S/390: DIGITS, HEX, VARGRAPHIC.

- The concatenation operator, '||', used with literals or alphanumeric columns.
- The following aggregate functions: COUNT, MIN, MAX, SUM, and AVG.
- The following expressions can appear in conditions: CASE, NULLIF, and COALESCE.
- Date, time, and timestamp literals of several different formats. For details, see *SQL Translator Support for Date, Time, and Timestamp Fields* on page 19-15.
- All requests that contain ANY, SOME, and ALL that do not contain =ALL, <>ANY, and <>SOME.
- =ALL, <>ANY, and <>SOME for requests that are candidates for Dialect Translation if the RDBMS engine supports quantified subqueries.
- The special registers USER, CURRENT\_DATE, CURRENT\_TIME, CURRENT\_TIMESTAMP, CURRENT\_EDASQLVERSION, and CURRENT\_TIMEZONE.
- NULL and NOT NULL predicates.
- LIKE and NOT LIKE predicates.
- IN and NOT IN predicates.
- Date and time arithmetic.
- EXISTS and NOT EXISTS predicates.
- GROUP BY clause expressed using explicit column names.
- ORDER BY clause expressed using explicit column names or column numbers.
- FOR FETCH ONLY feature to circumvent record locking.
- Continental Decimal Notation (CDN) when the CDN variable is set.
- National language support (NLS).
- Temporary defined columns. (Permanent defined columns are supported—that is, those that are defined in the iWay Dynamic Catalog or in the Master File.)

## **Reference**

### **Unsupported SQL Statements**

SQL Translation Services does not support the following:

- More than 15 joins per SELECT. This limit is set by SQL; FOCUS supports up to 16 joins.
- ALIAS names in Master Files and the use of formatting options to format output.
- Unique truncations of column names.
- Correlated subqueries for DML Generation.
- =ALL, <>ANY, and <>SOME for DML Generation.

## **Reference**

### **SQL Translator Reserved Words**

The following words may not be used as field names in a Master File that is used with the SQL Translator:

- ALL
- COUNT
- SUM
- MAX
- MIN
- AVG
- CURRENT
- DISTINCT
- USER

# Using SQL Translator Commands

The SQL command may be used to report from any supported data source or set of data sources. Standard TABLE phrases for formatting reports can be appended to the SQL statements to take advantage of a wide range of report preparation options.

**Note:** If you need to join data sources for your request, you have two options: you can use the JOIN command before you issue any SQL statements, or you can use the WHERE predicate in the SQL SELECT statement to join the required files dynamically. See *SQL Joins* on page 19-8.

## Syntax

### How to Use SQL Translator Commands

```
SQL
sql statement;
[ECHO|FILE]
[TABLE phrases]
END
```

where:

**SQL**

Is the SQL command identifier, which invokes the SQL Translator.

**Note:** The SQL command components must appear in the order represented above.

**sql statement**

Is a supported SQL statement. The statement must be terminated by a semicolon; it can continue for more than one line. For details, see *Supported SQL Statements* on page 19-2.

Within the SQL statement, field names are limited to 48 characters (an ANSI standard Level 2 limitation); view names generated through the SQL CREATE VIEW statement are limited to 18 characters; subqueries can be nested up to 15 levels deep. Correlated subqueries are not supported by FOCUS and other non-relational data sources.

**ECHO**

Are optional debugging phrases that capture the generated TABLE request. These options are placed after the SQL statement.

**FILE [name]**

Writes the translated TABLE phrases to the named procedure. If you do not supply a file name, a default name is assigned when the request runs; the file is then deleted.

**TABLE phrases**

Optional TABLE formatting phrases. For additional information, see *TABLE Formatting Phrases in SQL Requests* on page 19-6.

**END or QUIT**

Required to terminate procedure.

## Example      Using SQL Translator Commands

The following request contains an SQL statement and TABLE formatting commands:

```
SQL
SELECT BODYTYPE, AVG(MPG), SUM(SALES)
FROM CAR
WHERE RETAIL_COST > 5000
GROUP BY BODYTYPE;
TABLE HEADING CENTER
"AVERAGE MPG AND TOTAL SALES PER BODYTYPE"
END
```

## Reference      TABLE Formatting Phrases in SQL Requests

You can include TABLE formatting phrases in an SQL request, subject to the following rules:

- You can use TABLE formatting phrases with SELECT and UNION only.
- You must introduce the formatting phrases with the word TABLE.
- You may specify headings and footings, describe actions with an ON phrase, or use the ON TABLE SET command. Additionally, you can use ON TABLE HOLD or ON TABLE PCHOLD to create an extract file. You can also specify READLIMIT and RECORDLIMIT tests.

For details on headings and footings, see Chapter 9, *Customizing Tabular Reports*.

For details on ON TABLE HOLD or ON TABLE PCHOLD, see Chapter 11, *Saving and Reusing Report Output*.

- You cannot specify additional display fields, ACROSS fields, WHERE or IF criteria (other than READLIMIT or RECORDLIMIT tests), or calculated values; BY phrases are ignored.

## Automatic Passthru

If you are accessing a relational data source, the SQL Translator automatically generates a Direct SQL Passthru request when the SQL submitted contains valid syntax for the RDBMS being accessed. As a result, the SQL code is not processed by FOCUS, but instead is sent directly to the RDBMS. Column names displayed in the report will contain the RDBMS table's column names, which correspond with the Master File's ALIAS values. If this behavior is not desirable, you can issue the following command in the SQL query:

```
SQL
SET APT = OFF;
sql statement
END
```

## The SQL SELECT Statement

The basic SQL SELECT statement translates into one or more TABLE PRINT or TABLE SUM commands, depending on whether individual field display or aggregation is applied in the request. For details, see Chapter 2, *Displaying Report Data*.

**Note:** If you are accessing a relational data source, Automatic Passthru may be invoked as described in *Automatic Passthru* on page 19-6.

The SQL statement SELECT \* translates to a PRINT of every field in the Master File and uses all of the fields of the Cartesian product. This is a quick way to display a file, provided it fits in a reasonable number of screens for display, or provided you use ON TABLE HOLD or ON TABLE PCHOLD to retain retrieved data in a file for reuse. For details, see Chapter 11, *Saving and Reusing Report Output*.

SQL functions (such as COUNT, SUM, MAX, MIN, AVG) are supported in SELECT lists and HAVING conditions. Expressions may be used as function arguments.

The function COUNT (\*) translates to a count of the number of records produced by printing all fields in the Master File. This is the same as counting all rows in the Cartesian product that results from a SELECT on all fields.

Whenever possible, expressions in the SQL WHERE predicate are translated into corresponding WHERE criteria in the TABLE request. Expressions in SELECT lists generate virtual fields. The SQL HAVING clauses also translate into corresponding WHERE TOTAL criteria in the TABLE request. The SQL LIKE operator is translated directly into the corresponding LIKE operator in the WHERE criteria of the TABLE request. For details on record selection in TABLE requests, see Chapter 5, *Selecting Records for Your Report*.

Only subqueries based on equality, when the WHERE expression is compared to a subquery using an equal (=) sign, are supported. For example: WHERE field = (SELECT ...).

The SQL UNION operator translates to a TABLE request that creates a HOLD file for each data source specified, followed by a MATCH command with option HOLD OLD-OR-NEW, which combines records from both the first (old) data source and the second (new) data source. For details, see Chapter 14, *Merging Data Sources*.

For related information see *Supported SQL Statements* on page 19-2 and *How to Use SQL Translator Commands* on page 19-5.

## SQL Joins

When performing SQL joins, the formats of the joined fields must be the same. Join fields need not be indexed, and non-equijoins are supported.

Recursive, outer, and inner joins are supported. Inner join is the default.

### Syntax

#### How to Create an Inner JOIN

Two syntax variations are supported for inner joins.

##### Variation 1

```
SQL
SELECT fieldlist FROM file1 [alias1], file2 [alias2]
[WHERE where_condition];
END
```

##### Variation 2

```
SQL
SELECT fieldlist FROM file1 [alias1] INNER JOIN file2 [alias2]
ON join_condition [INNER JOIN ...]
[WHERE where_condition];
END
```

where:

*fieldlist*

Identifies which fields are to be retrieved from which data sources.

Joined fields in the SQL WHERE predicate must be qualified if the names are not unique; that is, they must be specified with their corresponding file names (or file aliases). For example:

```
{file1|alias1}.field1, {file2|alias2}.field2
```

*FROM*

Introduces the data sources to be joined.

*file1, file2*

Are the data sources to be joined.

*alias1, alias2*

Are optional alternate names for the data sources to be joined.

*where\_condition*

Is an optional selection condition for the joined answer set. Joined rows that do not satisfy this condition are eliminated from the returned answer set. If omitted in Variation 1, the answer set is the Cartesian product of the two data sources.

*join\_condition*

Is the join condition.

## Syntax

### How to Create an Outer JOIN

```
SQL
SELECT fieldlist FROM file1 {LEFT|RIGHT} JOIN file2
ON join_condition [{LEFT|RIGHT} JOIN ...]
WHERE where_condition
END
```

where:

*fieldlist*

Identifies which fields are to be retrieved from which data sources.

Joined fields in the SQL WHERE predicate must be qualified if the names are not unique; that is, they must be specified with their corresponding file names (or file aliases). For example:

```
{file1|alias1}.field1, {file2|alias2}.field2
```

FROM

Introduces the data sources to be joined.

*file1, file2*

Are the data sources to be joined.

*alias1, alias2*

Are optional alternate names for the data sources to be joined.

*join\_condition\_*

Is the join condition. The condition must specify equality. For example:

```
T1.A=T2.B.
```

*where\_condition*

Is an optional selection condition for the joined answer set. Joined rows that do not satisfy this condition are eliminated from the returned answer set.

## Reference

### JOIN Name Assignments From SQL Translator

Note that joins issued by the SQL Translator are assigned names in the format

*SQLJNMnn*

where:

*SQLJNM*

Is the SQL Translator join prefix.

*nn*

Is a number between 01 and 16 assigned in the order in which the joins are created. (FOCUS supports up to a maximum of 16 joins.)

For example, the first join will have the AS name SQLJNM01, the second join will be named SQLJNM02, and so on, up to SQLJNM16.

All joins are automatically created and cleared by the SQL Translator. No user-specified JOINS are affected.

## **Example**      **Using Qualified Field Names in SQL Joins**

In the following example, T.A and U.B are qualified field names:

```
SQL
SELECT T.A, T.B
FROM T, U
WHERE T.A = U.B;
END
```

## **Example**      **Using Recursive SQL Joins**

In the following statement, A and B are aliases for the same data source, CAR. The output from CAR is pairs of B values that have the same A values:

```
SQL
SELECT A.SEATS, B.SEATS
FROM CAR A, CAR B
WHERE A.MODEL = B.MODEL;
END
```

Note that all field names in the SELECT clause must be unique or qualified.

## **Reference**      **SQL JOIN Considerations**

- In standard SQL, WHERE field='a' selects records where the field has the value 'a' or 'A'. The SQL Translator is case-sensitive and returns only the exact value requested ('a' only).
- The SQL comparison operators ANY, SOME, and ALL are supported, with the exception of =ALL, <>ANY, and <>SOME.
- Sub-selects are not supported in HAVING conditions.
- In a multi-segment structure, parent segments are omitted from reports if no instances of their descendant segments exist. This is an inner join.
- The SQL Translator applies optimization techniques when constructing joins. These are described in *Index Optimized Retrieval* on page 19-20.



## SQL CREATE TABLE and INSERT Commands

SQL Translator supports the commands CREATE TABLE and INSERT INTO table:

- CREATE TABLE is used to create a new data source table. CREATE TABLE only generates single-segment Master Files.
- INSERT INTO is used to insert a row or block of rows into a table or view. Single-record insert with actual data values is supported.

These commands enable you to create tables to enhance reporting efficiency.

**Note:** When applications are enabled, the Master File and data source are written to the APPHOLD directory. When applications are disabled, the Master File and data source are written to the TEMP directory.

### Reference

#### Usage Notes for CREATE TABLE and INSERT Commands

- According to normal SQL data definition syntax, each CREATE TABLE or INSERT statement must terminate with a semicolon.
- The CREATE TABLE command supports the INTEGER, SMALLINT, FLOAT, CHARACTER, DATE, TIME, TIMESTAMP, DECIMAL, DOUBLE PRECISION and REAL data types. Decimals are rounded in the DOUBLE PRECISION and REAL data types.
- When using the CREATE TABLE and INSERT commands, the data type FLOAT should be declared with a precision and used in an INSERT without the 'E' designation. This requires the entire value to be specified without an exponent.
- The CHECK and DEFAULT options are not supported with the CREATE TABLE command.

### Example

#### Creating a Table With Single-Record Insert

The following example shows the use of the single-record insert, creating the table U with one record:

```

-* Single-record insert example.
-*
SQL
CREATE TABLE U (A INT, B CHAR(6), C CHAR(6), X INT, Y INT);
END
SQL
INSERT INTO U (A,B,C,X,Y) VALUES (10, '123456','654321', 10, 15);
END

```

## SQL CREATE VIEW and DROP VIEW Commands

A view is a transient object that inherits most of the characteristics of a table. Like a table, a view is composed of rows and columns:

- **CREATE VIEW** is used to create views. (Note that **CREATE VIEW** does not put the view in the system catalog.)
- **DROP VIEW** is used to explicitly remove transient tables and views from the environment.

### Tip:

To use the view, issue a **SELECT** from the view. You cannot issue a **TABLE** request against the view because the view is not extracted as a physical **FOCUS** data source. To create a **HOLD** file for extracted data, specify **ON TABLE HOLD** after the SQL statements. For details on creating **HOLD** files, see Chapter 11, *Saving and Reusing Report Output*.

## Syntax

### How to Create a View

The SQL Translator supports the following SQL statement:

```
CREATE VIEW viewname AS subquery ;
```

where:

*viewname*

Is the name of the view.

*subquery*

Is a **SELECT** statement that nests inside: a **WHERE**, **HAVING**, or **SELECT** clause of another **SELECT**; an **UPDATE**, **DELETE**, or **INSERT** statement; another subquery.

## Example

### Creating and Reporting From an SQL View

The following example creates a view named **XYZ**:

```
SQL
CREATE VIEW XYZ
 AS SELECT CAR, MODEL
 FROM CAR;
END
```

To report from the view, issue:

```
SQL
SELECT CAR, MODEL
FROM XYZ;
END
```

According to normal SQL data definition syntax, each **CREATE VIEW** statement must terminate with a semicolon.

**Example      Dropping an SQL View**

The following request removes the XYZ view:

```
SQL
 DROP VIEW XYZ;
END
```

**Cartesian Product Style Answer Sets**

The SQL Translator automatically generates Cartesian product style answer sets unless you explicitly turn this feature off. However, it is advisable to leave the CARTESIAN setting on since turning it off does not comply with ANSI standards. For details on the SET CARTESIAN command, see Chapter 14, *Merging Data Sources*.

**Continental Decimal Notation (CDN)**

Continental decimal notation displays numbers using a comma to mark the decimal position and periods for separating significant digits into groups of three. This notation is available for SQL Translator requests.

**Example      Using CDN to Separate Digits**

The following example creates a column defined as 1.2 + SEATS:

```
SET CDN=ON
SQL
 SELECT SEATS + 1,2
 FROM CAR;
END
```

## Specifying Field Names in SQL Requests

In an SQL request, you can specify fields using:

- **Delimited identifiers.** A field name may contain (but not begin with) the symbols ., #, @, \_, and \$. You must enclose such field names in double quotation marks when referring to them in requests.
- **Qualified field names.** You can qualify a field name with file and file alias names. File alias names are described in the discussion of joins in *SQL Joins* on page 19-8. See the *Describing Data* manual for more information on qualified field names.
- **Field names with embedded blanks and special characters.** A SELECT list can specify field names with embedded blanks or other special characters; you must enclose such field names in double quotation marks. Special characters are any characters not listed in *Delimited Identifiers* and not contained in the national character set of the FOCUS environment installed.

### *Example*

#### Specifying a Field Name With a Delimited Identifier

The following field identifier can be included in a request:

```
"COUNTRY.NAME"
```

### *Example*

#### Qualifying a Delimited Field Name

To qualify the delimited field name COUNTRY.NAME with its file name, use the syntax:

```
CAR. "COUNTRY.NAME"
```

## SQL UNION, INTERSECT, and EXCEPT Operators

The SQL UNION, INTERSECT, and EXCEPT operators generate MATCH logic. The number of files that can participate is determined by the MATCH limit. UNION with parentheses is supported.

- SELECT A UNION SELECT B retrieves rows in A or B or both. (This is equivalent to the MATCH phrase OLD-OR-NEW.)
- INTERSECT retrieves rows in both A and B. (This is equivalent to the MATCH phrase OLD-AND-NEW.)
- EXCEPT retrieves rows in A, but not B. (This is equivalent to the MATCH phrase OLD-NOT-NEW.)

Match logic merges the contents of your data sources. For details, see Chapter 14, *Merging Data Sources*.

## Numeric Constants, Literals, Expressions, and Functions

The SQL SELECT list, WHERE predicate, and HAVING clause can include numeric constants, literals enclosed in single quotation marks, expressions, and any scalar functions.

Internally, a virtual field is created for each such item in the SELECT list; the value of the virtual field is provided in the answer set.

## SQL Translator Support for Date, Time, and Timestamp Fields

Several new data types have been defined to the SQL Translator to support date-time fields in the WHERE predicate or field list of a SELECT statement.

In addition, time or timestamp columns can be defined in relational or FOCUS data sources. These are accessible to the translator. Values can be entered using INSERT and UPDATE statements and displayed in SELECT statements.

Time or timestamp data items (columns or literals) can be compared in conditions. Time values or timestamp values can be added or subtracted from each other, with the result being the number of seconds difference. Expressions of the form T + 2 HOURS or TS + 5 YEARS are allowed. These expressions will be translated to calls to the date-time functions described in the *Using Functions* manual.

All date formats for actual and virtual fields in the Master File are converted to the form YYYYMMDD. If you specify a format that lacks any component, the SQL Translator supplies a default value for the missing component. To specify a portion of a date, such as the month, use a virtual field with an alphanumeric format.

**Reference**      **SQL Translator Support for Date, Time, and Timestamp Fields**

In the following chart, fff represents the second to three decimal places (milliseconds) and ffffff represents the second to six decimal places (microseconds).

The following formats are allowed as input to the Translator:

| Format                   | USAGE Attribute in Master File | Date Components            |
|--------------------------|--------------------------------|----------------------------|
| Date                     | YYMD                           | YYYY-MM-DD                 |
| Hour                     | HH                             | HH                         |
| Hour through minute      | HHI                            | HH.MM                      |
| Hour through second      | HHIS                           | HH.MM.SS                   |
| Hour through millisecond | HHISs                          | HH.MM.SS.fff               |
| Hour through microsecond | HHISsm                         | HH.MM.SS.ffffff            |
| Year through hour        | HYYMDH                         | YYYY-MM-DD HH              |
| Year through minute      | HYYMDI                         | YYYY-MM-DD HH.MM           |
| Year through second      | HYYMDS                         | YYYY-MM-DD HH.MM.SS        |
| Year through millisecond | HYYMDS                         | YYYY-MM-DD HH.MM.SS.fff    |
| Year through microsecond | HYYMDm                         | YYYY-MM-DD HH.MM.SS.ffffff |

**Note:**

- Time information may be given to the hour, minute, second, or fraction of a second.
- The separator within date information may be either a hyphen or a slash.
- The separator within time information must be a colon.
- The separator between date and time information must be a space.

## Extracting Date-Time Components Using the SQL Translator

The SQL Translator supports several functions that return components from date-time values. You can use the `EXTRACT` statement to extract components.

You can also use the `TRIM` subroutine to remove leading and or trailing patterns from date, time, and timestamp values. For details, see the *Using Functions* manual.

### Syntax

### Date, Time, and Timestamp Functions Accepted by the SQL Translator

The following functions return date-time components as integer values. Assume *x* is a date-time value:

| Function                    | Return Value |
|-----------------------------|--------------|
| <code>YEAR(x)</code>        | year         |
| <code>MONTH(x)</code>       | month number |
| <code>DAY(x)</code>         | day number   |
| <code>HOURL(x)</code>       | hour         |
| <code>MINUTE(x)</code>      | minute       |
| <code>SECOND(x)</code>      | second       |
| <code>MILLISECOND(x)</code> | millisecond  |
| <code>MICROSECOND(x)</code> | microsecond  |

### Example

### Using SQL Translator Date, Time, and Timestamp Functions

Using the timestamp column `TS` whose value is '1999-11-23 07:32:16.123456':

```
YEAR(TS) = 1999
MONTH(TS) = 11
DAY(TS) = 23
HOUR(TS) = 7
MINUTE(TS) = 32
SECOND(TS) = 16
MILLISECOND(TS) = 123
MICROSECOND(TS) = 123456
```

**Example**                      **Using SQL Translator Date, Time, and Timestamp Functions in a SELECT Statement**

Assume that a FOCUS data source called VIDEOTR2 includes a Date-time field named TRANSDATE.

```
SQL
SELECT TRANSDATE,
YEAR(TRANSDATE), MONTH(TRANSDATE),
MINUTE(TRANSDATE)
FROM VIDEOTR2;
FILE VIDSQ END
```

The SQL Translator produces the following virtual fields for functions, followed by a TABLE request to display the output:

```
-SET &SQLVARFN=&FOCFIELDNAME ;
SET FIELDNAME=NOTRUNC
SET COUNTWIDTH=ON
JOIN CLEAR SQLJNM*
END

DEFINE FILE VIDEOTR2
SQLDEF01/I5 = HPART(TRANSDATE,'YEAR','I5'); SQLDEF02/I5 = INT(SQLDEF01);
SQLDEF03/I3 = HPART(TRANSDATE,'MONTH','I3'); SQLDEF04/I3 = INT(SQLDEF03);
SQLDEF05/I3 = HPART(TRANSDATE,'MINUTE','I3'); END

TABLEF FILE VIDEOTR2
PRINT TRANSDATE SQLDEF02 SQLDEF04 SQLDEF05 ON TABLE SET CARTESIAN ON
ON TABLE SET ASNAMES ON
ON TABLE SET HOLDLIST PRINTONLY
END
```

The output is:

| TRANSDATE        | SQLDEF02 | SQLDEF04 | SQLDEF05 |
|------------------|----------|----------|----------|
| 1999/06/20 04:14 | 1999     | 6        | 14       |
| 1991/06/27 02:45 | 1991     | 6        | 45       |
| 1996/06/21 01:16 | 1996     | 6        | 16       |
| 1991/06/21 07:11 | 1991     | 6        | 11       |
| 1991/06/20 05:15 | 1991     | 6        | 15       |
| 1999/06/26 12:34 | 1999     | 6        | 34       |
| 1919/06/26 05:45 | 1919     | 6        | 45       |
| 1991/06/21 01:10 | 1991     | 6        | 10       |
| 1991/06/19 07:18 | 1991     | 6        | 18       |
| 1991/06/19 04:11 | 1991     | 6        | 11       |
| 1998/10/03 02:41 | 1998     | 10       | 41       |
| 1991/06/25 01:19 | 1991     | 6        | 19       |
| 1986/02/05 03:30 | 1986     | 2        | 30       |
| 1991/06/24 04:43 | 1991     | 6        | 43       |
| 1991/06/24 02:08 | 1991     | 6        | 8        |
| 1999/10/06 02:51 | 1999     | 10       | 51       |
| 1991/06/25 01:17 | 1991     | 6        | 17       |



## Syntax

### Using the SQL Translator EXTRACT Function to Extract Date-Time Components

You can use the following ANSI standard function to extract date-time components as integer values:

```
EXTRACT(component FROM value)
```

where:

*component*

Is one of the following: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, MILLISECOND, or MICROSECOND.

*value*

Is a date-time, DATE, TIME, or TIMESTAMP field, constant or expression.

For example, the following are equivalent:

```
EXTRACT(YEAR FROM TS)
YEAR(TS)
```

## Example

### Using the EXTRACT Function

```
SELECT D. EXTRACT(YEAR FROM D), EXTRACT(MONTH FROM D),
 EXTRACT(DAY FROM D) FROM T
```

This request produces rows similar to the following:

|            |      |   |   |
|------------|------|---|---|
| 1999-01-01 | 1999 | 1 | 1 |
| 2000-03-03 | 2000 | 3 | 3 |

## Index Optimized Retrieval

The SQL Translator improves query performance by generating optimized code that enables the underlying retrieval engine to access the selected records directly, without scanning all segment instances.

For more information about index optimization and optimized join statements, see the iWay documentation for your platform.

## Optimized Joins

The SQL Translator accepts joins in SQL syntax. SQL language joins have no implied direction; that is, the concepts of host and cross-referenced files do not exist in SQL.

The SQL Translator analyzes each join in order to identify an efficient implementation, as reflected in the following illustration. First, it assigns costs to the candidate joins in the query:

- Cost = 1 for an equijoin to a field that can participate as a cross-referenced field according to FOCUS join rules. This is common in queries against relational tables with equijoin predicates in the WHERE clause.
- Cost = 16 for an equijoin to a field that cannot participate as a cross-referenced field according to FOCUS join rules.
- Cost = 256 for a non-equijoin or an unrestricted Cartesian product.

The Translator then uses these costs to build a join structure for the query. The order of the tables listed in the FROM clause of the query influences the first two phases of the join analysis:

1. If there are cost=1 joins from the first table referenced in the FROM clause to the second, from the second table to the third, and so on, the Translator joins the tables in the order specified in the query. If not, it goes on to Phase 2.
2. If Phase 1 fails to generate an acceptable join structure, the Translator attempts to generate a join structure without joining any table TO a table that precedes it in the FROM clause. Therefore, this phase always makes the first table referenced in the query the host table. If there is no cost=1 join between two tables, or if using one would require changing the table order, the Translator abandons Phase 2 and implements Phase 3.
3. The Translator generates the join structure by using the lowest cost joins first and then picking from the more expensive ones as necessary. This sorting process may change the order in which tables are joined. The efficiency of the join that this procedure generates is somewhat dependent on the relative sizes of the tables being joined.

If the analysis results in joining TO a table that cannot participate as a cross-referenced file according to FOCUS rules (because it lacks an index, for example), the Translator generates code to build an indexed HOLD file and implements the join with this file. However, the HOLD file does not participate in the analysis of join order.

# TABLEF Optimization

To improve performance, the SQL Translator can be set to generate FOCUS TABLEF commands instead of TABLE commands. You can take advantage of this optimization using the SET SQLTOPTTF command (SQL Translator OPTimization TableF). For related information, see Chapter 15, *Improving Report Processing*.

## Syntax

### How to Improve Performance Using SQLTOPTTF

```
SET SQLTOPTTF = {ON|OFF}
```

where:

ON

Causes TABLEF commands to be generated when possible (for example, if there is no join or GROUP BY phrase). This is the default.

OFF

Causes TABLE commands to be generated.

## SQL INSERT, UPDATE, and DELETE Commands

The SQL INSERT, UPDATE, and DELETE commands enable SQL users to manipulate and modify data:

- The INSERT statement is used to introduce new rows into an existing table.
- The DELETE statement removes a row or combination of rows from a table.
- The UPDATE statement permits users to update a row or group of rows in a table.

You can issue an SQL INSERT, UPDATE, or DELETE command against one segment instance (row) at a time. When you issue one of these commands against a multi-segment Master File:

- All fields referenced in the command must be on a single path through the file structure.
- The command must explicitly specify (in the WHERE predicate) every key value from the root to the target segment instance, and this combination of key values must uniquely identify one segment instance (row) to be affected by the command.

If you are modifying every field in the row, you can omit the list of field names from the command.

- The SQL Translator does not support subqueries, such as:

```
INSERT...INTO...SELECT...FROM...
```

Although each INSERT, UPDATE, or DELETE command can specify only one row, referential integrity constraints may produce the following additional modifications to the data source:

- If you delete a segment instance that has descendant segment instances (children), the children are automatically deleted.
- If you insert a segment for which parent segments are missing, the parent segments are automatically created.

---

## APPENDIX A

# Master Files and Diagrams

### Topics:

- Creating Sample Data Sources
- The EMPLOYEE Data Source
- The JOBFIL Data Source
- The EDUCFIL Data Source
- The SALES Data Source
- The PROD Data Source
- The CAR Data Source
- The LEDGER Data Source
- The FINANCE Data Source
- The REGION Data Source
- The COURSES Data Source
- The EMPDATA Data Source
- The EXPERSON Data Source
- The TRAINING Data Source
- The PAYHIST File
- The COMASTER File
- The VideoTrk and MOVIES Data Sources
- The VIDEOTR2 Data Source
- The Gotham Grinds Data Sources

This appendix contains data source descriptions and structure diagrams for the examples used throughout the documentation.

# Creating Sample Data Sources

You can create the sample data sources on your user ID by executing the procedures specified below. These FOCEXECs are supplied with FOCUS. If they are not available to you or if they produce error messages, contact your systems administrator.

To create these files, first make sure you have read access to the Master Files.

| Data Source                                        | Load Procedure Name                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EMPLOYEE,<br>EDUCFILE, and<br>JOBFILE              | Under CMS enter:<br><br>EX EMPTEST<br><br>Under MVS, enter:<br><br>EX EMPTSO<br><br>These FOCEXECs also test the data sources by generating sample reports. If you are using Hot Screen, remember to press either Enter or the PF3 key after each report. If the EMPLOYEE, EDUCFILE, and JOBFILE data sources already exist on your user ID, the FOCEXEC will replace the data sources with new copies. This FOCEXEC assumes that the high-level qualifier for the FOCUS data sources will be the same as the high-level qualifier for the MASTER PDS that was unloaded from the tape. |
| SALES<br>PROD                                      | EX SALES<br>EX PROD                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| CAR                                                | none (created automatically during installation)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| LEDGER<br>FINANCE<br>REGION<br>COURSES<br>EXPERSON | EX LEDGER<br>EX FINANCE<br>EX REGION<br>EX COURSES<br>EX EXPERSON                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| EMPDATA<br>TRAINING                                | EX LOADEMP<br>EX LOADTRAI                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| PAYHIST                                            | none (PAYHIST DATA is a sequential data source and is allocated during the installation process)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| COMASTER                                           | none (COMASTER is used for debugging other Master Files)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| VideoTrk and<br>MOVIES                             | EX LOADVTRK                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| VIDEOTR2                                           | EX LOADVID2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Gotham Grinds                                      | EX LOADGG                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## The EMPLOYEE Data Source

The EMPLOYEE data source contains data about a company's employees. Its segments are:

- EMPINFO, which contains employee IDs, names, and positions.
- FUNDTRAN, which specifies employees' direct deposit accounts. This segment is unique.
- PAYINFO, which contains the employee's salary history.
- ADDRESS, which contains employees' home and bank addresses.
- SALINFO, which contains data on employees' monthly pay.
- DEDUCT, which contains data on monthly pay deductions.

The EMPLOYEE data source also contains cross-referenced segments belonging to the JOBFIL and EDUCFIL files, described later in this appendix. The segments are:

- JOBSEG (from JOBFIL), which describes the job positions held by each employee.
- SECSEG (from JOBFIL), which lists the skills required by each position.
- SKILLSEG (from JOBFIL), which specifies the security clearance needed for each job position.
- ATTNDSEG (from EDUCFIL), which lists the dates that employees attended in-house courses.
- COURSEG (from EDUCFIL), which lists the courses that the employees attended.

## The EMPLOYEE Master File

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO, SEGTYPE=S1
 FIELDNAME=EMP_ID, ALIAS=EID, FORMAT=A9, $
 FIELDNAME=LAST_NAME, ALIAS=LN, FORMAT=A15, $
 FIELDNAME=FIRST_NAME, ALIAS=FN, FORMAT=A10, $
 FIELDNAME=HIRE_DATE, ALIAS=HDT, FORMAT=I6YMD, $
 FIELDNAME=DEPARTMENT, ALIAS=DPT, FORMAT=A10, $
 FIELDNAME=CURR_SAL, ALIAS=CSAL, FORMAT=D12.2M, $
 FIELDNAME=CURR_JOBCODE, ALIAS=CJC, FORMAT=A3, $
 FIELDNAME=ED_HRS, ALIAS=OJT, FORMAT=F6.2, $
SEGNAME=FUNDTRAN, SEGTYPE=U, PARENT=EMPINFO
 FIELDNAME=BANK_NAME, ALIAS=BN, FORMAT=A20, $
 FIELDNAME=BANK_CODE, ALIAS=BC, FORMAT=I6S, $
 FIELDNAME=BANK_ACCT, ALIAS=BA, FORMAT=I9S, $
 FIELDNAME=EFFECT_DATE, ALIAS=EDATE, FORMAT=I6YMD, $
SEGNAME=PAYINFO, SEGTYPE=SH1, PARENT=EMPINFO
 FIELDNAME=DAT_INC, ALIAS=DI, FORMAT=I6YMD, $
 FIELDNAME=PCT_INC, ALIAS=PI, FORMAT=F6.2, $
 FIELDNAME=SALARY, ALIAS=SAL, FORMAT=D12.2M, $
 FIELDNAME=JOBCODE, ALIAS=JBC, FORMAT=A3, $
SEGNAME=ADDRESS, SEGTYPE=S1, PARENT=EMPINFO
 FIELDNAME=TYPE, ALIAS=AT, FORMAT=A4, $
 FIELDNAME=ADDRESS_LN1, ALIAS=LN1, FORMAT=A20, $
 FIELDNAME=ADDRESS_LN2, ALIAS=LN2, FORMAT=A20, $
 FIELDNAME=ADDRESS_LN3, ALIAS=LN3, FORMAT=A20, $
 FIELDNAME=ACCTNUMBER, ALIAS=ANO, FORMAT=I9L, $
SEGNAME=SALINFO, SEGTYPE=SH1, PARENT=EMPINFO
 FIELDNAME=PAY_DATE, ALIAS=PD, FORMAT=I6YMD, $
 FIELDNAME=GROSS, ALIAS=MO_PAY, FORMAT=D12.2M, $
SEGNAME=DEDUCT, SEGTYPE=S1, PARENT=SALINFO
 FIELDNAME=DED_CODE, ALIAS=DC, FORMAT=A4, $
 FIELDNAME=DED_AMT, ALIAS=DA, FORMAT=D12.2M, $
SEGNAME=JOBSEG, SEGTYPE=KU, PARENT=PAYINFO, CRFILE=JOBFILE, CRKEY=JOBCODE,$
SEGNAME=SECSEG, SEGTYPE=KLU, PARENT=JOBSEG, CRFILE=JOBFILE,$
SEGNAME=SKILLSEG, SEGTYPE=KL, PARENT=JOBSEG, CRFILE=JOBFILE,$
SEGNAME=ATTNDSEG, SEGTYPE=KM, PARENT=EMPINFO, CRFILE=EDUCFILE, CRKEY=EMP_ID,$
SEGNAME=COURSESEG, SEGTYPE=KLU, PARENT=ATTNDSEG, CRFILE=EDUCFILE,$
```



# The EMPLOYEE Structure Diagram

STRUCTURE OF FOCUS      FILE EMPLOYEE ON 09/15/00 AT 10.16.27

```

EMPINFO
01 S1

*EMP_ID **
*LAST_NAME **
*FIRST_NAME **
*HIRE_DATE **
* **

I
+-----+-----+-----+-----+
I I I I I
I FUNDTRAN I PAYINFO I ADDRESS I SALINFO I ATTNDSEG
02 I U 03 I SH1 07 I S1 08 I SH1 10 I KM

*BANK_NAME * * *DAT_INC ** *TYPE ** *PAY_DATE ** :DATE_ATTEND :
*BANK_CODE * * *PCT_INC ** *ADDRESS_LN1 ** *GROSS ** :EMP_ID :K
*BANK_ACCT * * *SALARY ** *ADDRESS_LN2 ** * ** : :
*EFFECT_DATE * * *JOBCODE ** *ADDRESS_LN3 ** * ** : :
* * * ** * ** * ** : :

I
I
I
I JOBSEG
04 I KU

:JOBCODE :K
:JOB_DESC :
: :
: :
: :
: :

I JOBFILE
I
+-----+-----+
I I
I SECSEG I SKILLSEG
05 I KLU 06 I KL

:SEC_CLEAR : :SKILLS :
: : :SKILL_DESC :
: : : :
: : : :
: : : :
: : : :

JOBFILE *****
JOBFILE EDUCFILE

```

# The JOBFIL Data Source

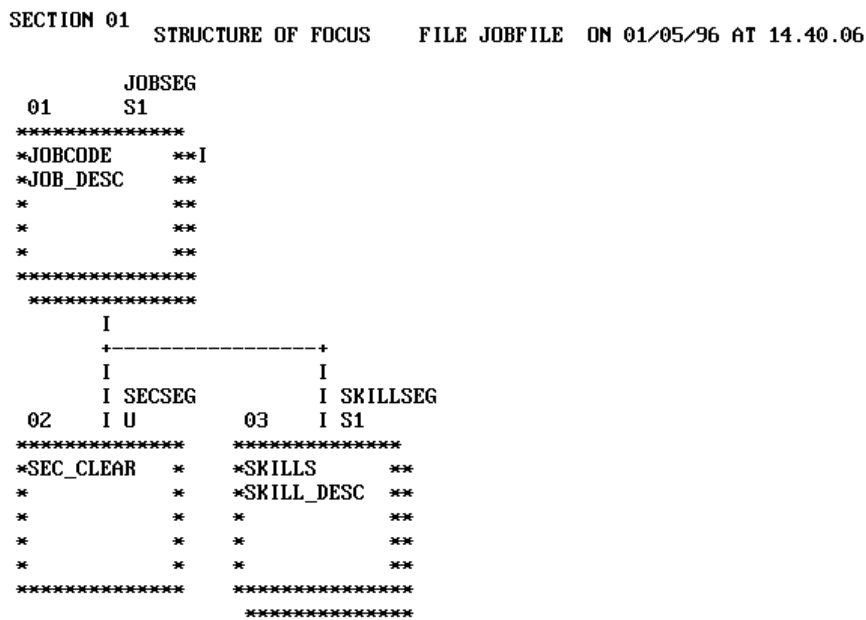
The JOBFIL data source contains information on a company’s job positions. Its segments are:

- JOBSEG describes what each position is. The field JOBCODE in this segment is indexed.
- SKILLSEG lists the skills required by each position.
- SECSEG specifies the security clearance needed, if any. This segment is unique.

## The JOBFIL Master File

```
FILENAME=JOBFIL ,SUFFIX=FOC
SEGNAME=JOBSEG ,SEGTYPE=S1
FIELD=JOBCODE ,ALIAS=JC ,USAGE=A3 ,INDEX=1,$
FIELD=JOB_DESC ,ALIAS=JD ,USAGE=A25 ,
SEGNAME=SKILLSEG ,SEGTYPE=S1 ,PARENT=JOBSEG
FIELD=SKILLS ,ALIAS= ,USAGE=A4 ,
FIELD=SKILL_DESC ,ALIAS=SD ,USAGE=A30 ,
SEGNAME=SECSEG ,SEGTYPE=U ,PARENT=JOBSEG
FIELD=SEC_CLEAR ,ALIAS=SC ,USAGE=A6 ,
```

## The JOBFIL Structure Diagram



# The EDUCFILE Data Source

The EDUCFILE data source contains data on a company's in-house courses. Its segments are:

- COURSEG contains data on each course.
- ATTNDSEG specifies which employees attended the courses. Both fields in the segment are key fields. The field EMP\_ID in this segment is indexed.

## The EDUCFILE Master File

```

FILENAME=EDUCFILE ,SUFFIX=FOC
SEGNAME=COURSESEG ,SEGTYPE=S1
 FIELD=COURSE_CODE ,ALIAS=CC ,USAGE=A6 ,,$
 FIELD=COURSE_NAME ,ALIAS=CD ,USAGE=A30 ,,$
SEGNAME=ATTNDSEG ,SEGTYPE=SH2 ,PARENT=COURSESEG
 FIELD=DATE_ATTEND ,ALIAS=DA ,USAGE=I6YMD ,,$
 FIELD=EMP_ID ,ALIAS=EID ,USAGE=A9 ,INDEX=I,$

```

## The EDUCFILE Structure Diagram

```

SECTION 01
 STRUCTURE OF FOCUS FILE EDUCFILE ON 01/05/96 AT 14.45.44
 COURSEG
01 S1

*COURSE_CODE **
*COURSE_NAME **
* **
* **
* **

 I
 I
 I
 I ATTNDSEG
02 I SH2

*DATE_ATTEND **
*EMP_ID **I
* **
* **
* **


```

## The SALES Data Source

The SALES data source records sales data for a dairy company (or a store chain). Its segments are:

- STOR\_SEG lists the stores buying the products.
- DAT\_SEG contains the dates of inventory.
- PRODUCT contains sales data for each product on each date. Note the following about fields in this segment:
  - The PROD\_CODE field is indexed.
  - The RETURNS and DAMAGED fields have the MISSING=ON attribute.

## The SALES Master File

```
FILENAME=KSALES, SUFFIX=FOC,

SEGNAME=STOR_SEG, SEGTYPE=S1,
 FIELDNAME=STORE_CODE, ALIAS=SNO, FORMAT=A3, $
 FIELDNAME=CITY, ALIAS=CTY, FORMAT=A15, $
 FIELDNAME=AREA, ALIAS=LOC, FORMAT=A1, $

SEGNAME=DATE_SEG, PARENT=STOR_SEG, SEGTYPE=SH1,
 FIELDNAME=DATE, ALIAS=DTE, FORMAT=A4MD, $

SEGNAME=PRODUCT, PARENT=DATE_SEG, SEGTYPE=S1,
 FIELDNAME=PROD_CODE, ALIAS=PCODE, FORMAT=A3, FIELDTYPE=1, $
 FIELDNAME=UNIT_SOLD, ALIAS=SOLD, FORMAT=I5, $
 FIELDNAME=RETAIL_PRICE, ALIAS=RP, FORMAT=D5.2M, $
 FIELDNAME=DELIVER_AMT, ALIAS=SHIP, FORMAT=I5, $
 FIELDNAME=OPENING_AMT, ALIAS=INV, FORMAT=I5, $
 FIELDNAME=RETURNS, ALIAS=RTN, FORMAT=I3, MISSING=ON, $
 FIELDNAME=DAMAGED, ALIAS=BAD, FORMAT=I3, MISSING=ON, $
```

# The SALES Structure Diagram

SECTION 01

STRUCTURE OF FOCUS

FILE SALES ON 01/05/96 AT 14.50.28

```

 STOR_SEG
01 S1

*STORE_CODE **
*CITY **
*AREA **
* **
* **

 I
 I
 I
 I DATE_SEG
02 I SH1

*DATE **
* **
* **
* **
* **

 I
 I
 I
 I PRODUCT
03 I S1

*PROD_CODE **I
*UNIT_SOLD **
*RETAIL_PRICE**
*DELIVER_AMT **
* **

 I
 I

```

# The PROD Data Source

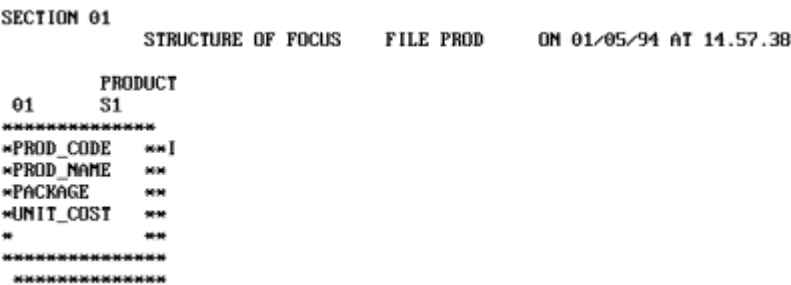
The PROD data source lists products sold by a dairy company. It consists of one segment, PRODUCT. The field PROD\_CODE is indexed.

## The PROD Master File

```
FILE=KPROD, SUFFIX=FOC,

SEGMENT=PRODUCT, SEGTYPE=S1,
 FIELDNAME=PROD_CODE, ALIAS=PCODE, FORMAT=A3, FIELDTYPE=I, $
 FIELDNAME=PROD_NAME, ALIAS=ITEM, FORMAT=A15, $
 FIELDNAME=PACKAGE, ALIAS=SIZE, FORMAT=A12, $
 FIELDNAME=UNIT_COST, ALIAS=COST, FORMAT=D5.2M, $
```

## The PROD Structure Diagram



# The CAR Data Source

The CAR data source contains specifications and sales information for rare cars. Its segments are:

- ORIGIN lists the country that manufactures the car. The field COUNTRY is indexed.
- COMP contains the car name.
- CARREC contains the car model.
- BODY lists the body type, seats, dealer and retail costs, and units sold.
- SPECS lists car specifications. This segment is unique.
- WARRANT lists the type of warranty.
- EQUIP lists standard equipment.

The aliases in the CAR Master File are specified without the ALIAS keyword.

## The CAR Master File

```

FILENAME=CAR,SUFFIX=FOC
SEGNAME=ORIGIN,SEGTYPE=S1
 FIELDNAME=COUNTRY,COUNTRY,A10,FIELDTYPE=I,$
SEGNAME=COMP,SEGTYPE=S1,PARENT=ORIGIN
 FIELDNAME=CAR,CARS,A16,$
SEGNAME=CARREC,SEGTYPE=S1,PARENT=COMP
 FIELDNAME=MODEL,MODEL,A24,$
SEGNAME=BODY,SEGTYPE=S1,PARENT=CARREC
 FIELDNAME=BODYTYPE,TYPE,A12,$
 FIELDNAME=SEATS,SEAT,I3,$
 FIELDNAME=DEALER_COST,DCOST,D7,$
 FIELDNAME=RETAIL_COST,RCOST,D7,$
 FIELDNAME=SALES,UNITS,I6,$
SEGNAME=SPECS,SEGTYPE=U,PARENT=BODY
 FIELDNAME=LENGTH,LEN,D5,$
 FIELDNAME=WIDTH,WIDTH,D5,$
 FIELDNAME=HEIGHT,HEIGHT,D5,$
 FIELDNAME=WEIGHT,WEIGHT,D6,$
 FIELDNAME=WHEELBASE,BASE,D6.1,$
 FIELDNAME=FUEL_CAP,FUEL,D6.1,$
 FIELDNAME=BHP,POWER,D6,$
 FIELDNAME=RPM,RPM,I5,$
 FIELDNAME=MPG,MILES,D6,$
 FIELDNAME=ACCEL,SECONDS,D6,$
SEGNAME=WARRANT,SEGTYPE=S1,PARENT=COMP
 FIELDNAME=WARRANTY,WARR,A40,$
SEGNAME=EQUIP,SEGTYPE=S1,PARENT=COMP
 FIELDNAME=STANDARD,EQUIP.A40.$

```

SECTION 01                      STRUCTURE OF FOCUS                      FILE CAR                      ON 01/05/96 AT 14.59.29

```

ORIGIN
01 S1

**COUNTRY **I
** **
** **
** **
** **

I
I
I
I COMP
02 I S1

**CAR **
** **
** **
** **
** **

I
+-----+-----+-----+
I I I
I CARREC I WARIANT I EQUIP
03 I S1 06 I S1 07 I S1

**MODEL ** **WARRANTY ** **STANDARD **
** ** ** ** ** **
** ** ** ** ** **
** ** ** ** ** **
** ** ** ** ** **

I
I
I
I BODY
04 I S1

**BODYTYPE **
**SEATS **
**DEALER_COST **
**RETAIL_COST **
** **

I
I
I
I SPECS
05 I U

**LENGTH **
**WIDTH **
**HEIGHT **
**WEIGHT **
** **

```



# The LEDGER Data Source

The LEDGER data source lists accounting information. It consists of one segment, TOP. This data source is specified primarily for FML examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

## The LEDGER Master File

```
FILENAME=LEDGER, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
 FIELDNAME=YEAR , , FORMAT=A4, $
 FIELDNAME=ACCOUNT, , FORMAT=A4, $
 FIELDNAME=AMOUNT , , FORMAT=I5C,$
```

## The LEDGER Structure Diagram

```
SECTION 01 STRUCTURE OF FOCUS FILE LEDGER ON 01/05/96 AT 15.07.56

 TOP
01 S2

*YEAR **
*ACCOUNT **
*AMOUNT **
* **
* **


```

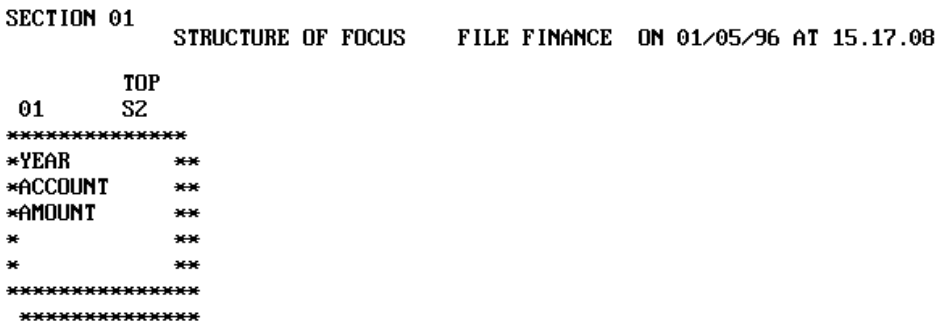
# The FINANCE Data Source

The FINANCE data source contains financial information for balance sheets. It consists of one segment, TOP. This data source is specified primarily for FML examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

## The FINANCE Master File

```
FILENAME=FINANCE, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
 FIELDNAME=YEAR , , FORMAT=A4, $
 FIELDNAME=ACCOUNT, , FORMAT=A4, $
 FIELDNAME=AMOUNT , , FORMAT=D12C,$
```

## The FINANCE Structure Diagram



## The REGION Data Source

The REGION data source lists account information for the east and west regions of the country. It consists of one segment, TOP. This data source is specified primarily for FML examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

### The REGION Master File

```
FILENAME=REGION, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S1,$
 FIELDNAME=ACCOUNT , , FORMAT=A4, $
 FIELDNAME=E_ACTUAL, , FORMAT=I5C,$
 FIELDNAME=E_BUDGET, , FORMAT=I5C,$
 FIELDNAME=W_ACTUAL, , FORMAT=I5C,$
 FIELDNAME=W_BUDGET, , FORMAT=I5C,$
```

### The REGION Structure Diagram

```
SECTION 01 STRUCTURE OF FOCUS FILE REGION ON 01/05/96 AT 15.18.48

 TOP
01 S1

*ACCOUNT **
*e_ACTUAL **
*e_BUDGET **
*w_ACTUAL **
* **


```

# The COURSES Data Source

The COURSES data source describes education courses. It consists of one segment, CRSESEG1. The field DESCRIPTION has a format of TEXT (TX).

## The COURSES Master File

```
FILENAME=COURSES, SUFFIX=FOC, $
SEGNAME=CRSESEG1, SEGTYPE=S1, $
 FIELDNAME=COURSE_CODE, ALIAS=CC, FORMAT=A6, FIELDTYPE=I, $
 FIELDNAME=COURSE_NAME, ALIAS=CN, FORMAT=A30, $
 FIELDNAME=DURATION, ALIAS=DAYS, FORMAT=I3, $
 FIELDNAME=DESCRIPTION, ALIAS=CDESC, FORMAT=TX50, $
```

## The COURSES Structure Diagram

```
SECTION 01 STRUCTURE OF FOCUS FILE COURSES ON 01/05/94 AT 15.20.59

 CRSESEG1
01 S1

* COURSE_CODE **I
* COURSE_NAME **
* DURATION **
* DESCRIPTION **T
* **


```

## The EMPDATA Data Source

The EMPDATA data source contains organizational data about a company's employees. It consists of one segment, EMPDATA. Note the following:

- The PIN field is indexed.
- The AREA field is a temporary one.

## The EMPDATA Master File

```

FILENAME=EMPDATA, SUFFIX=FOC
SEGNAME=EMPDATA, SEGTYPE=S1
 FIELDNAME=PIN, ALIAS=ID, FORMAT=A9, INDEX=I, $
 FIELDNAME=LASTNAME, ALIAS=LN, FORMAT=A15, $
 FIELDNAME=FIRSTNAME, ALIAS=FN, FORMAT=A10, $
 FIELDNAME=MIDINITIAL, ALIAS=MI, FORMAT=A1, $
 FIELDNAME=DIU, ALIAS=CDIU, FORMAT=A4, $
 FIELDNAME=DEPT, ALIAS=CDEPT, FORMAT=A20, $
 FIELDNAME=JOBCLASS, ALIAS=CJCLAS, FORMAT=A8, $
 FIELDNAME=TITLE, ALIAS=CFUNC, FORMAT=A20, $
 FIELDNAME=SALARY, ALIAS=CSAL, FORMAT=D12.2M, $
 FIELDNAME=HIREDATE, ALIAS=HDAT, FORMAT=YMD, $
$
DEFINE AREA/A13=DECODE DIU (NE 'NORTH EASTERN' SE 'SOUTH EASTERN'
CE 'CENTRAL' WE 'WESTERN' CORP 'CORPORATE' ELSE 'INVALID AREA');$

```

## The EMPDATA Structure Diagram

```

SECTION 01 STRUCTURE OF FOCUS FILE EMPDATA ON 01/05/96 AT 14.49.09

 EMPDATA
01 S1

*PIN **I
*LASTNAME **
*FIRSTNAME **
*MIDINITIAL **
* **


```

## The EXPERSON Data Source

The EXPERSON data source contains personal data about individual employees. It consists of one segment, ONESEG.

### The EXPERSON Master File

```
FILE=EXPERSON ,SUFFIX=FOC
SEGMENT=ONESEG, $
 FIELDNAME=SOC_SEC_NO ,ALIAS=SSN ,USAGE=A9 ,,$
 FIELDNAME=FIRST_NAME ,ALIAS=FN ,USAGE=A9 ,,$
 FIELDNAME=LAST_NAME ,ALIAS=LN ,USAGE=A10 ,,$
 FIELDNAME=AGE ,ALIAS=YEARS ,USAGE=I2 ,,$
 FIELDNAME=SEX ,ALIAS= ,USAGE=A1 ,,$
 FIELDNAME=MARITAL_STAT ,ALIAS=MS ,USAGE=A1 ,,$
 FIELDNAME=NO_DEP ,ALIAS=NDP ,USAGE=I3 ,,$
 FIELDNAME=DEGREE ,ALIAS= ,USAGE=A3 ,,$
 FIELDNAME=NO_CARS ,ALIAS=CARS ,USAGE=I3 ,,$
 FIELDNAME=ADDRESS ,ALIAS= ,USAGE=A14 ,,$
 FIELDNAME=CITY ,ALIAS= ,USAGE=A10 ,,$
 FIELDNAME=WAGE ,ALIAS=PAY ,USAGE=D10.2SM ,,$
 FIELDNAME=CATEGORY ,ALIAS=STATUS ,USAGE=A1 ,,$
 FIELDNAME=SKILL_CODE ,ALIAS=SKILLS ,USAGE=A5 ,,$
 FIELDNAME=DEPT_CODE ,ALIAS=WHERE ,USAGE=A4 ,,$
 FIELDNAME=TEL_EXT ,ALIAS=EXT ,USAGE=I4 ,,$
 FIELDNAME=DATE_EMP ,ALIAS=BASE_DATE ,USAGE=I6YMTD ,,$
 FIELDNAME=MULTIPLIER ,ALIAS=RATIO ,USAGE=D5.3 ,,$
```

### The EXPERSON Structure Diagram

```
SECTION 01 STRUCTURE OF FOCUS FILE EXPERSON ON 01/05/96 AT 14.50.58

 ONESEG
01 S1

*SOC_SEC_NO **
*FIRST_NAME **
*LAST_NAME **
*AGE **
* **


```

## The TRAINING Data Source

The TRAINING data source contains training course data for employees. It consists of one segment, TRAINING. Note the following:

- The PIN field is indexed.
- The EXPENSES, GRADE, and LOCATION fields have the MISSING=ON attribute.

## The TRAINING Master File

```

FILENAME=TRAINING, SUFFIX=FOC
SEGNAME=TRAINING, SEGTYPE=SH3
FIELDNAME=PIN, ALIAS=ID, FORMAT=A9, INDEX=I, $
FIELDNAME=COURSESTART, ALIAS=CSTART, FORMAT=YMD, $
FIELDNAME=COURSECODE, ALIAS=CCOD, FORMAT=A7, $
FIELDNAME=EXPENSES, ALIAS=COST, FORMAT=D8.2, MISSING=ON,$
FIELDNAME=GRADE, ALIAS=GRA, FORMAT=A2, MISSING=ON,$
FIELDNAME=LOCATION, ALIAS=LOC, FORMAT=A6, MISSING=ON,$

```

## The TRAINING Structure Diagram

```

SECTION 01
 STRUCTURE OF FOCUS FILE TRAINING ON 12/12/94 AT 14.51.28

 TRAINING
01 SH3

*PIN **I
*COURSESTART **
*COURSECODE **
*EXPENSES **
* **


```

# The PAYHIST File

The PAYHIST data source contains the employees’ salary history. It consists of one segment, PAYSEG. The SUFFIX attribute indicates that the data file is a fixed-format sequential file.

## The PAYHIST Master File

```
FILENAME=PAYHIST, SUFFIX=FIX
SEGMENT=PAYSEG,$
 FIELDNAME=SOC_SEC_NO, ALIAS=SSN, USAGE=A9, ACTUAL=A9,$
 FIELDNAME=DATE_OF_IN, ALIAS=INCDATE, USAGE=I6YMTD, ACTUAL=A6,$
 FIELDNAME=AMT_OF_INC, ALIAS=RAISE, USAGE=D6.2, ACTUAL=A10,$
 FIELDNAME=PCT_INC, ALIAS=, USAGE=D6.2, ACTUAL=A6,$
 FIELDNAME=NEW_SAL, ALIAS=CURR_SAL, USAGE=D10.2, ACTUAL=A11,$
 FIELDNAME=FILL, ALIAS=, USAGE=A38, ACTUAL=A38,$
```

## The PAYHIST Structure Diagram

```
SECTION 01 STRUCTURE OF FIX FILE PAYHIST ON 01/05/96 AT 14.51.59

 PAYSEG
01 S1

*SOC_SEC_NO **
*DATE_OF_IN **
*AMT_OF_INC **
*PCT_INC **
* **


```



## The COMASTER File

The COMASTER file is used to display the file structure and contents of each segment in a data source. Since COMASTER is used for debugging other Master Files, a corresponding FOCEXEC does not exist for the COMASTER file. Its segments are:

- FILEID lists file information.
- RECID lists segment information.
- FIELDID lists field information.
- DEFREC lists a description record.
- PASSREC lists read/write access.
- CRSEG lists cross-reference information for segments.
- ACCSEG lists DBA information.

## The COMASTER Master File

```

FILE=COMASTER, SUFFIX=COM,

SEGNAME=FILEID
FIELDNAME=FILENAME ,FILE ,A8 , ,,$
FIELDNAME=FILE SUFFIX ,SUFFIX ,A8 , ,,$

SEGNAME=RECID
FIELDNAME=SEGNAME ,SEGMENT ,A8 , ,,$
FIELDNAME=SEGTYPE ,SEGTYPE ,A4 , ,,$
FIELDNAME=SEGSIZE ,SEGSIZE ,14 , A4,$
FIELDNAME=PARENT ,PARENT ,A8 , ,,$
FIELDNAME=CRKEY ,CRKEY ,A66 , ,,$

SEGNAME=FIELDID
FIELDNAME=FIELDNAME ,FIELD ,A66 , ,,$
FIELDNAME=ALIAS ,SYNONYM ,A66 , ,,$
FIELDNAME=FORMAT ,USAGE ,A8 , ,,$
FIELDNAME=ACTUAL ,ACTUAL ,A8 , ,,$
FIELDNAME=AUTHORITY ,AUTHCODE ,A8 , ,,$
FIELDNAME=FIELDTYPE ,INDEX ,A8 , ,,$
FIELDNAME=TITLE ,TITLE ,A64 , ,,$
FIELDNAME=HELPMESSAGE ,MESSAGE ,A256 , ,,$
FIELDNAME=MISSING ,MISSING ,A4 , ,,$
FIELDNAME=ACCEPTS ,ACCEPTABLE ,A255 , ,,$
FIELDNAME=RESERVED ,RESERVED ,A44 , ,,$

SEGNAME=DEFREC
FIELDNAME=DEFINITION ,DESCRIPTION ,A44 , ,,$

SEGNAME=PASSREC,PARENT=FILEID
FIELDNAME=READ/WRITE ,RW ,A32 , ,,$

SEGNAME=CRSEG,PARENT=RECID
FIELDNAME=CRFILENAME ,CRFILE ,A8 , ,,$
FIELDNAME=CRSEGNAME ,CRSEGMENT ,A8 , ,,$
FIELDNAME=ENCRYPT ,ENCRYPT ,A4 , ,,$

SEGNAME=ACCSEG,PARENT=DEFREC
FIELDNAME=DBA ,DBA ,A8 , ,,$
FIELDNAME=DBAFILE , ,A8 , ,,$
FIELDNAME=USER ,PASS ,A8 , ,,$
FIELDNAME=ACCESS ,ACCESS ,A8 , ,,$
FIELDNAME=RESTRICT ,RESTRICT ,A8 , ,,$
FIELDNAME=NAME ,NAME ,A66 , ,,$
FIELDNAME=VALUE ,VALUE ,A80 , ,,$

```

# The COMASTER Structure Diagram

## SECTION 01

STRUCTURE OF EXTERNAL FILE COMASTER ON 12/12/94 AT 14.53.38

```

 FILEID
01 S1

*FILENAME **
*FILE SUFFIX **
* **
* **
* **

 I
 +-----+
 I I
 I RECID I PASSREC
02 I N 07 I N
***** *****
*SEGNAME ** *READ/WRITE **
*SEGTYP ** * **
*SEGSIZE ** * **
*PARENT ** * **
* ** * **
***** *****
***** *****
 I
 +-----+
 I I
 I FIELDID I CRSEG
03 I N 06 I N
***** *****
*FIELDNAME ** *CRFILENAME **
*ALIAS ** *CRSEGNAME **
*FORMAT ** *ENCRYPT **
*ACTUAL ** * **
* ** * **
***** *****
***** *****
 I
 I
 I
 I DEFREC
04 I N

*DEFINITION **
* **
* **
* **
* **

 I
 I
 I
 I ACCSEG
05 I N

*DBA **
*DBA FILE **
*USER **
*ACCESS **
* **


```

## The VideoTrk and MOVIES Data Sources

The VideoTrk data source tracks customer, rental, and purchase information for a video rental business. It can be joined to the MOVIES data source. VideoTrk and MOVIES are used in examples that illustrate the use of the Maintain facility.

### VideoTrk Master File

```
FILENAME=VIDEOTRK, SUFFIX=FOC
SEGNAME=CUST, SEGTYPE=S1
 FIELDNAME=CUSTID, ALIAS=CIN, FORMAT=A4, $
 FIELDNAME=LASTNAME, ALIAS=LN, FORMAT=A15, $
 FIELDNAME=FIRSTNAME, ALIAS=FN, FORMAT=A10, $
 FIELDNAME=EXPDATE, ALIAS=EXDAT, FORMAT=YMD, $
 FIELDNAME=PHONE, ALIAS=TEL, FORMAT=A10, $
 FIELDNAME=STREET, ALIAS=STR, FORMAT=A20, $
 FIELDNAME=CITY, ALIAS=CITY, FORMAT=A20, $
 FIELDNAME=STATE, ALIAS=PROV, FORMAT=A4, $
 FIELDNAME=ZIP, ALIAS=POSTAL_CODE, FORMAT=A9, $
SEGNAME=TRANSDAT, SEGTYPE=SH1, PARENT=CUST
 FIELDNAME=TRANSDATE, ALIAS=OUTDATE, FORMAT=YMD, $
SEGNAME=SALES, SEGTYPE=S2, PARENT=TRANSDAT
 FIELDNAME=PRODCODE, ALIAS=PCOD, FORMAT=A6, $
 FIELDNAME=TRANSCODE, ALIAS=TCOD, FORMAT=I3, $
 FIELDNAME=QUANTITY, ALIAS=NO, FORMAT=I3S, $
 FIELDNAME=TRANSTOT, ALIAS=TTOT, FORMAT=F7.2S, $
SEGNAME=RENTALS, SEGTYPE=S2, PARENT=TRANSDAT
 FIELDNAME=MOVIECODE, ALIAS=MCOD, FORMAT=A6, INDEX=I, $
 FIELDNAME=COPY, ALIAS=COPY, FORMAT=I2, $
 FIELDNAME=RETURNDATE, ALIAS=INDATE, FORMAT=YMD, $
 FIELDNAME=FEE, ALIAS=FEE, FORMAT=F5.2S, $
```

### MOVIES Master File

```
FILENAME=MOVIES, SUFFIX=FOC
SEGNAME=MOVINFO, SEGTYPE=S1
 FIELDNAME=MOVIECODE, ALIAS=MCOD, FORMAT=A6, INDEX=I, $
 FIELDNAME=TITLE, ALIAS=MTL, FORMAT=A39, $
 FIELDNAME=CATEGORY, ALIAS=CLASS, FORMAT=A8, $
 FIELDNAME=DIRECTOR, ALIAS=DIR, FORMAT=A17, $
 FIELDNAME=RATING, ALIAS=RTG, FORMAT=A4, $
 FIELDNAME=RELDATE, ALIAS=RDAT, FORMAT=YMD, $
 FIELDNAME=WHOLESALEPR, ALIAS=WPRC, FORMAT=F6.2, $
 FIELDNAME=LISTPR, ALIAS=LPRC, FORMAT=F6.2, $
 FIELDNAME=COPIES, ALIAS=NOC, FORMAT=I3, $
```

## VideoTrk Structure Diagram

SECTION 01

STRUCTURE OF FOCUS

FILE VIDEOTRK ON 05/21/99 AT 12.25.19

```

 CUST
01 S1

*CUSTID **
*LASTNAME **
*FIRSTNAME **
*EXPDATE **
* **

 I
 I
 I
 I TRANSDAT
02 I SH1

*TRANSDATE **
* **
* **
* **
* **

 I
 +-----+
 I I
 I SALES I RENTALS
03 I S2 04 I S2
***** *****
*PRODCODE ** *MOVIECODE **I
*TRANSCODE ** *COPY **
*QUANTITY ** *RETURNDATE **
*TRANSTOT ** *FEE **
* ** * **
***** *****


```

# MOVIES Structure Diagram

```
SECTION 01
STRUCTURE OF FOCUS FILE MOVIES ON 05/21/99 AT 12.26.05

MOVINFO
01 S1

*MOVIECODE **I
*TITLE **
*CATEGORY **
*DIRECTOR **
* **


```

# The VIDEOTR2 Data Source

The VIDEOTR2 data source tracks customer, rental, and purchase information for a video rental business. It is similar to VideoTrk but is a partitioned data source with both a Master and Access File and with a date-time field.

# The VIDEOTR2 Master File

```
FILENAME=VIDEOTR2, SUFFIX=FOC,
ACCESS=VIDEOACX, $
SEGNAME=CUST, SEGTYPE=S1
FIELDNAME=CUSTID, ALIAS=CIN, FORMAT=A4, $
FIELDNAME=LASTNAME, ALIAS=LN, FORMAT=A15, $
FIELDNAME=FIRSTNAME, ALIAS=FN, FORMAT=A10, $
FIELDNAME=EXPDATE, ALIAS=EXDAT, FORMAT=YMD, $
FIELDNAME=PHONE, ALIAS=TEL, FORMAT=A10, $
FIELDNAME=STREET, ALIAS=STR, FORMAT=A20, $
FIELDNAME=CITY, ALIAS=CITY, FORMAT=A20, $
FIELDNAME=STATE, ALIAS=PROV, FORMAT=A4, $
FIELDNAME=ZIP, ALIAS=POSTAL_CODE, FORMAT=A9, $
FIELDNAME=EMAIL, ALIAS=EMAIL, FORMAT=A18, $
SEGNAME=TRANSDAT, SEGTYPE=SH1, PARENT=CUST
FIELDNAME=TRANSDATE, ALIAS=OUTDATE, FORMAT=HYYYMDI, $
SEGNAME=SALES, SEGTYPE=S2, PARENT=TRANSDAT
FIELDNAME=TRANSCODE, ALIAS=TCOD, FORMAT=I3, $
FIELDNAME=QUANTITY, ALIAS=NO, FORMAT=I3S, $
FIELDNAME=TRANSTOT, ALIAS=TTOT, FORMAT=F7.2S, $
SEGNAME=RENTALS, SEGTYPE=S2, PARENT=TRANSDAT
FIELDNAME=MOVIECODE, ALIAS=MCOD, FORMAT=A6, INDEX=I, $
FIELDNAME=COPY, ALIAS=COPY, FORMAT=I2, $
FIELDNAME=RETURNDATE, ALIAS=INDATE, FORMAT=YMD, $
FIELDNAME=FEE, ALIAS=FEE, FORMAT=F5.2S, $
DEFINE DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
```

## The VIDEOTR2 Access File

On CMS,

```
MASTER VIDEOTR2
 DATANAME 'VIDPART1 FOCUS A'
 WHERE DATE EQ 1991;

 DATANAME 'VIDPART2 FOCUS A'
 WHERE DATE FROM 1996 TO 1998;

 DATANAME 'VIDPART3 FOCUS A'
 WHERE DATE FROM 1999 TO 2000;
```

On MVS, the data set names include your user ID as the high-level qualifier:

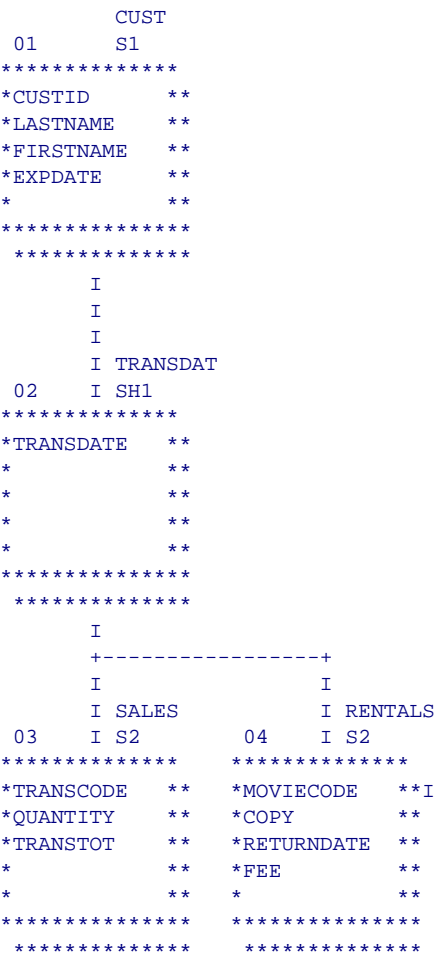
```
MASTER VIDEOTR2
 DATANAME userid.VIDPART1.FOCUS
 WHERE DATE EQ 1991;

 DATANAME userid.VIDPART2.FOCUS
 WHERE DATE FROM 1996 TO 1998;

 DATANAME userid.VIDPART2.FOCUS
 WHERE DATE FROM 1999 TO 2000;
```

# The VIDEOTR2 Structure Diagram

STRUCTURE OF FOCUS      FILE VIDEOTR2 ON 09/27/00 AT 16.45.48





# The Gotham Grinds Data Sources

Gotham Grinds is a group of data sources that contain information about a specialty items company.

## The GGDEMOG Data Source

The GGDEMOG data source contains demographic information about the customers of Gotham Grinds, a company that sells specialty items like coffee, gourmet snacks, and gifts. It consists of one segment, DEMOG01.

### The GGDEMOG Master File

```
FILENAME=GGDEMOG, SUFFIX=FOC
SEGNAME=DEMOG01, SEGTYPE=S1
 FIELD=ST, ALIAS=E02, FORMAT=A02, INDEX=I, TITLE='State', DESC='State', $
 FIELD=HH, ALIAS=E03, FORMAT=I09, TITLE='Number of Households',
 DESC='Number of Households', $
 FIELD=AUGHHS298, ALIAS=E04, FORMAT=I09, TITLE='Average Household Size',
 DESC='Average Household Size', $
 FIELD=MEDHHI98, ALIAS=E05, FORMAT=I09, TITLE='Median Household Income',
 DESC='Median Household Income', $
 FIELD=AUGHHI98, ALIAS=E06, FORMAT=I09, TITLE='Average Household Income',
 DESC='Average Household Income', $
 FIELD=MALEPOP98, ALIAS=E07, FORMAT=I09, TITLE='Male Population',
 DESC='Male Population', $
 FIELD=FEMPOP98, ALIAS=E08, FORMAT=I09, TITLE='Female Population',
 DESC='Female Population', $
 FIELD=P15T01998, ALIAS=E09, FORMAT=I09, TITLE='15 to 19',
 DESC='Population 15 to 19 years old', $
 FIELD=P20T02998, ALIAS=E10, FORMAT=I09, TITLE='20 to 29',
 DESC='Population 20 to 29 years old', $
 FIELD=P30T04998, ALIAS=E11, FORMAT=I09, TITLE='30 to 49',
 DESC='Population 30 to 49 years old', $
 FIELD=P50T06498, ALIAS=E12, FORMAT=I09, TITLE='50 to 64',
 DESC='Population 50 to 64 years old', $
 FIELD=P65OVR98, ALIAS=E13, FORMAT=I09, TITLE='65 and over',
 DESC='Population 65 and over', $
```

## The GGDEMOG Structure Diagram

```

NUMBER OF ERRORS= 0
NUMBER OF SEGMENTS= 1 (REAL= 1 VIRTUAL= 0)
NUMBER OF FIELDS= 12 INDEXES= 1 FILES= 1
TOTAL LENGTH OF ALL FIELDS= 101
SECTION 01
 STRUCTURE OF FOCUS FILE GGDEMOG ON 09/17/96 AT 12.18.05

 GGDEMOG
01 S1

*ST **I
*HH **
*AVGHHSZ98 **
*MEDHHI98 **
* **


```

## The GGORDER Data Source

The GGORDER data source contains order information for Gotham Grinds. It consists of two segments, ORDER01 and ORDER02, respectively.

### The GGORDER Master File

```

FILENAME=GGORDER, SUFFIX=FOC
SEGNAME=ORDER01, SEGTYPE=S1
 FIELD=ORDER_NUMBER, ALIAS=ORDNO1, FORMAT=I6, TITLE='Order,Number',
 DESC='Order Identification Number', $
 FIELD=ORDER_DATE, ALIAS=DATE, FORMAT=MDY, TITLE='Order,Date',
 DESC='Date order was placed', $
 FIELD=STORE_CODE, ALIAS=STCD, FORMAT=A5, TITLE='Store,Code',
 DESC='Store Identification Code (for order)', $
 FIELD=PRODUCT_CODE, ALIAS=PCD, FORMAT=A4, TITLE='Product,Code',
 DESC='Product Identification Code (for order)', $
 FIELD=QUANTITY, ALIAS=ORDUNITS, FORMAT=I8, TITLE='Ordered,Units',
 DESC='Quantity Ordered', $
SEGNAME=ORDER02, SEGTYPE=KU, PARENT=ORDER01, CRFILE=GGPRODS, CRKEY=PCD,
CRSEG=PRODS01 , $

```

## The GGORDER Structure Diagram

```

NUMBER OF ERRORS= 0
NUMBER OF SEGMENTS= 2 (REAL= 1 VIRTUAL= 1)
NUMBER OF FIELDS= 12 INDEXES= 0 FILES= 2
TOTAL LENGTH OF ALL FIELDS= 92
SECTION 01
 STRUCTURE OF FOCUS FILE GGORDER ON 09/17/96 AT 12.29.24

 GGORDER
01 S1

*ORDER_NUMBER**
*ORDER_DATE **
*STORE_CODE **
*PRODUCT_CODE**
* **

 I
 I
 I
 I ORDER02
02 I KU

:PRODUCT_ID :K
:PRODUCT_DESC:
:VENDOR_CODE :
:VENDOR_NAME :
: :
: :
: :

```

## The GGPRODS Data Source

The GGPRODS data source contains product information for Gotham Grinds. It consists of one segment, PRODS01.

### The GGPRODS Master File

```

FILENAME=GGPRODS, SUFFIX=FOC
SEGNAME=PRODS01, SEGTYPE=S1
FIELD=PRODUCT_ID, ALIAS=PCD, FORMAT=A4, INDEX=I, TITLE='Product,Code',
DESC='Product Identification Code', $
FIELD=PRODUCT_DESCRIPTION, ALIAS=PRODUCT, FORMAT=A16, TITLE='Product',
DESC='Product Name', $
FIELD=VENDOR_CODE, ALIAS=UCD, FORMAT=A4, INDEX=I, TITLE='Vendor ID',
DESC='Vendor Identification Code', $
FIELD=VENDOR_NAME, ALIAS=VENDOR, FORMAT=A23, TITLE='Vendor Name',
DESC='Vendor Name', $
FIELD=PACKAGE_TYPE, ALIAS=PACK, FORMAT=A7, TITLE='Package',
DESC='Packaging Style', $
FIELD=SIZE, ALIAS=SZ, FORMAT=I2, TITLE='Size', DESC='Package Size', $
FIELD=UNIT_PRICE, ALIAS=UNITPR, FORMAT=D7.2, TITLE='Unit,Price',
DESC='Price for one unit', $

```

## The GGPRODS Structure Diagram

```
NUMBER OF ERRORS= 0
NUMBER OF SEGMENTS= 1 (REAL= 1 VIRTUAL= 0)
NUMBER OF FIELDS= 7 INDEXES= 2 FILES= 1
TOTAL LENGTH OF ALL FIELDS= 63
SECTION 01
 STRUCTURE OF FOCUS FILE GGPRODS ON 09/17/96 AT 12.21.12

 GGPRODS
01 S1

*PRODUCT_ID **I
*VENDOR_CODE **I
*PRODUCT_DES>***
*VENDOR_NAME **
* **

```

## The GGSales Data Source

The GGSales data source contains sales information for Gotham Grinds. It consists of one segment, SALES01.

### The GGSales Master File

```
FILENAME=GGSales, SUFFIX=FOC
SEGNAME=SALES01, SEGTYPE=S1
FIELD=SEQ_NO, ALIAS=SEQ, FORMAT=I5, TITLE='Sequence#',
DESC='Sequence number in database',$
FIELD=CATEGORY, ALIAS=E02, FORMAT=A11, INDEX=I, TITLE='Category',
DESC='Product category',$
FIELD=PCD, ALIAS=E03, FORMAT=A04, INDEX=I, TITLE='Product ID',
DESC='Product Identification code (for sale)',$
FIELD=PRODUCT, ALIAS=E04, FORMAT=A16, TITLE='Product', DESC='Product name',$
FIELD=REGION, ALIAS=E05, FORMAT=A11, INDEX=I, TITLE='Region',
DESC='Region code',$
FIELD=ST, ALIAS=E06, FORMAT=A02, INDEX=I, TITLE='State', DESC='State',$
FIELD=CITY, ALIAS=E07, FORMAT=A20, TITLE='City', DESC='City',$
FIELD=STCD, ALIAS=E08, FORMAT=A05, INDEX=I, TITLE='Store ID',
DESC='Store identification code (for sale)',$
FIELD=DATE, ALIAS=E09, FORMAT=I8YYMD, TITLE='Date',
DESC='Date of sales report',$
FIELD=UNITS, ALIAS=E10, FORMAT=I08, TITLE='Unit Sales',
DESC='Number of units sold',$
FIELD=DOLLARS, ALIAS=E11, FORMAT=I08, TITLE='Dollar Sales',
DESC='Total dollar amount of reported sales',$
FIELD=BUDUNITS, ALIAS=E12, FORMAT=I08, TITLE='Budget Units',
DESC='Number of units budgeted',$
FIELD=BUDDOLLARS, ALIAS=E13, FORMAT=I08, TITLE='Budget Dollars',
DESC='Total sales quota in dollars',$
```

## The GGSales Structure Diagram

```

NUMBER OF ERRORS= 0
NUMBER OF SEGMENTS= 1 (REAL= 1 VIRTUAL= 0)
NUMBER OF FIELDS= 13 INDEXES= 5 FILES= 1
TOTAL LENGTH OF ALL FIELDS= 114
SECTION 01
 STRUCTURE OF FOCUS FILE GGSales ON 09/17/96 AT 12.22.19

 GGSales
01 S1

*SEQ_NO **
*CATEGORY **I
*PCD **I
*REGION **I
* **


```

## The GGSTORES Data Source

The GGSTORES data source contains information for each of Gotham Grinds' 12 stores in the United States. It consists of one segment, STORES01.

### The GGSTORES Master File

```

FILENAME=GGSTORES, SUFFIX=FOC
SEGNAME=STORES01, SEGTYPE=S1
 FIELD=STORE_CODE, ALIAS=E02, FORMAT=A05, INDEX=1, TITLE='Store ID',
 DESC='Franchisee ID Code', $
 FIELD=STORE_NAME, ALIAS=E03, FORMAT=A23, TITLE='Store Name',
 DESC='Store Name', $
 FIELD=ADDRESS1, ALIAS=E04, FORMAT=A19, TITLE='Contact',
 DESC='Franchisee Owner', $
 FIELD=ADDRESS2, ALIAS=E05, FORMAT=A31, TITLE='Address', DESC='Street Address', $
 FIELD=CITY, ALIAS=E06, FORMAT=A22, TITLE='City', DESC='City', $
 FIELD=STATE, ALIAS=E07, FORMAT=A02, INDEX=1, TITLE='State', DESC='State', $
 FIELD=ZIP, ALIAS=E08, FORMAT=A06, TITLE='Zip Code', DESC='Postal Code', $

```

## The GGSTORES Structure Diagram

```
NUMBER OF ERRORS= 0
NUMBER OF SEGMENTS= 1 (REAL= 1 VIRTUAL= 0)
NUMBER OF FIELDS= 7 INDEXES= 2 FILES= 1
TOTAL LENGTH OF ALL FIELDS= 108
```

SECTION 01

STRUCTURE OF FOCUS FILE GGSTORES ON 09/17/96 AT 12.23.09

```
 GGSTORES
01 S1

*STORE_CODE **I
*STATE_ **I
*STORE_NAME **
*ADDRESS1 **
* **


```

---

## APPENDIX B

# Error Messages

### Topics:

- Accessing Error Files
- Displaying Messages Online

If you need to see the text or explanation for any error message, you can display it online in your FOCUS session or find it in a standard FOCUS ERRORS file. All of the FOCUS error messages are stored in eight system ERRORS files.

## Accessing Error Files

For CMS, the ERRORS files are:

- FOT004                      ERRORS
- FOG004                      ERRORS
- FOM004                      ERRORS
- FOS004                      ERRORS
- FOA004                      ERRORS
- FSQXLT                      ERRORS
- FOCSTY                      ERRORS
- FOB004                      ERRORS

For MVS, these files are the following members in the ERRORS PDS:

- FOT004
- FOG004
- FOM004
- FOS004
- FOA004
- FSQXLT
- FOCSTY
- FOB004



## Displaying Messages Online

To display a message online, issue the following query command at the FOCUS command level

? *n*

where *n* is the message number.

The message number and text will display along with a detailed explanation of the message (if one exists). For example, issuing the following command

? 210

displays the following:

(FOC210)      THE DATA VALUE HAS A FORMAT ERROR:

An alphabetic character has been found where all numerical digits are required.

---

## APPENDIX C

# Syntax Summary

### Topics:

- TABLE Syntax Summary
- TABLEF Syntax Summary
- MATCH Syntax Summary
- FOR Syntax Summary

This appendix summarizes FOCUS reporting commands and options.

# TABLE Syntax Summary

The syntax of a TABLE request is:

```
DEFINE FILE filename [CLEAR|ADD] [SAVE|RETURN]
tempfield[/format] [WITH realfield] = expression;
tempfield[/format] REDEFINES qualifier.fieldname=expression;
.
.
.
END
TABLE FILE filename
HEADING [CENTER]
" text "
{display command} [SEG.]field [/R|/L|/C] [/format]
{display command} [prefixop.]field [/R|/L|/C] [/format]
[NOPRINT|AS 'title1,...,title5'] [AND|OVER] [obj2...obj1024]
[WITHIN field] [IN n]
COMPUTE field[/format] =expression; [AS 'title,...,title5'] [IN n]
[AND] ROW-TOTAL [/R|/L|/C] [/format] [AS 'name']
[AND] COLUMN-TOTAL [/R|/L|/C] [AS 'name']
ACROSS [HIGHEST] sortfield [IN-GROUPS-OF qty] ACROSS-TOTAL [AS 'name']
[COLUMNS col1 AND col2 ...] [NOPRINT|AS 'title1,...,title5']
BY [HIGHEST|LOWEST(n)]TOTAL [prefix_operator]{field|code_value}
RANKED BY {TOP|HIGHEST|LOWEST} [n]sortfield
[IN-GROUPS-OF qty [TILES [TOP m]] [AS 'heading']]
[NOPRINT|AS 'title1,...,title5']
ON sortfield {SUBTOTAL|SUB-TOTAL|SUMMARIZE|RECOMPUTE} [MULTILINES]
[AS 'name'] [field1, field2,...] [WHEN expression;...]
ON sortfield {RECAP|COMPUTE} field[/format] =expression; [WHEN expression;...]
ON sfld RECAP fld1[/fmt]=FORECAST(fld2, intvl, npredct, '{MOVAVE|EXPAVE}',npnt);
ON sfld RECAP fld1[/fmt]=FORECAST(fld2, intvl, npredct, 'REGRESS');
WHERE [TOTAL] expression
WHERE RECORDLIMIT EQ n
WHERE READLIMIT EQ n
IF [TOTAL] field relation value [OR value...]
ON TABLE SET parameter value
ON TABLE HOLD [VIA program] [AS name] [FORMAT format] [MISSING {ON|OFF}]
ON TABLE PCHOLD [AS name] [FORMAT format] [MISSING {ON|OFF}]
ON TABLE SAVE [AS name] [FORMAT format] [MISSING {ON|OFF}]
ON TABLE SAVB [AS name] [FORMAT format] [MISSING {ON|OFF}]
ON TABLE NOTOTAL
ON TABLE COLUMN-TOTAL [/R|/L|/C] [AS 'name'] fieldname
ON TABLE ROW-TOTAL [/R|/L|/C] [format] [AS 'name'] fieldname
FOOTING [CENTER] [BOTTOM]
" text "
MORE
FILE file2
[IF field relation value [OR value...]|WHERE expression]
{END|RUN|QUIT}
```

# TABLEF Syntax Summary

The syntax of a TABLEF request is:

```
TABLEF FILE filename
HEADING [CENTER]
" text "

{display command} [SEG.]field [/R|/L|/C] [/format]
{display command} [prefixop.]field [/R|/L|/C] [/format]
 [NOPRINT|AS 'title1,...title5'] [AND|OVER] [obj2...obj495]
 [IN n]

COMPUTE field [/format]=expression; [AS 'title1,...title5']
[AND] ROW-TOTAL [AND] COLUMN-TOTAL

BY [HIGHEST] keyfieldn [NOPRINT]

ON keyfield option1 [AND] option2. . . .

WHERE [TOTAL] expression

IF [TOTAL] field relation value [OR value. . .]

ON TABLE SET parameter value

ON TABLE HOLD [VIA program] [AS name] [FORMAT format] [MISSING {ON|OFF}]
ON TABLE PCHOLD [AS name] [FORMAT format] [MISSING {ON|OFF}]
ON TABLE SAVE [AS name] [FORMAT format] [MISSING {ON|OFF}]
ON TABLE SAVB [AS name] [FORMAT format] [MISSING {ON|OFF}]

ON TABLE NOTOTAL
ON TABLE COLUMN-TOTAL fieldname
ON TABLE ROW-TOTAL fieldname

FOOTING [CENTER] [BOTTOM]
" text "

{END|RUN|QUIT}
```

## Note:

- Prefix operators for TABLEF can be: AVE., ASQ., MAX., MIN., PCT., RPCT., PCT.CNT., FST., LST., CNT., SUM., or TOT. TABLEF requests cannot use prefix operators PCT.CNT., RPCT., and TOT.

## MATCH Syntax Summary

The syntax of a MATCH request is:

```
MATCH FILE filename (the OLD file)

report request

BY field1 [AS sortfield]

MORE
FILE file3
subrequest

RUN
.
.
.

FILE filename2 (the NEW file)

report request

BY field1 [AS sortfield1]
.
.
.

[AFTER MATCH HOLD [AS filename] [FORMAT FOCUS] {matchtype}]

MORE
FILE file4
subrequest

END
```

where:

*matchtype*

Can be any of the following:

```
OLD
NEW
OLD-NOT-NEW
NEW-NOT-OLD
OLD-AND-NEW
OLD-OR-NEW
OLD-NOR-NEW
```

# FOR Syntax Summary

The formal syntax of the FOR statement is

```
FOR fieldname [NOPRINT]
row
[OVER row]
.
.
.
.
END
```

where:

*row*

Can be any of the following:

```
tag [OR tag...][options]
[fieldname]
DATA n,[n,...] $
DATA PICKUP [FROM filename] tag [LABEL label] [AS 'text']
RECAP name[/format]=expression;
BAR [AS 'character'] [OVER]
"text"
PAGE-BREAK [OVER]
```

*tag*

Can be any of the following:

```
value [OR value...]
value TO value
```

*option*

Can be any of the following:

```
AS 'text'
NOPRINT

[LABEL label]

WHEN EXISTS
NOPRINT

[POST [TO filename]]
```

---

## APPENDIX D

# Writing User-Coded Programs to Create HOLD Files

### Topic:

- Arguments Used in Calls to Programs That Create HOLD Files

HOLD files can be created by a user-coded program. This enables you to use the FOCUS report writer to obtain records from any FOCUS-readable data source and write the records to another data source for use by an external program. This feature is most useful when an external program requires an internal format or arrangement of data other than those already provided with the HOLD command formats (for example, FORMAT FOCUS, LOTUS, SQL).

FOCUS collects records from the report request and passes them, one at a time, to the user program.

## Arguments Used in Calls to Programs That Create HOLD Files

The program is called with the following arguments:

- RECNO is the record number in the HOLD file. The format is integer.
- LEN is the length of this record in the HOLD file. The format is integer.
- DDNAME is the name given in the HOLD AS phrase. The format is A8.
- RECORD is the record of data in the HOLD file. The format is Annnn (the maximum record length is 4096).
- RETCOD is the return code. The format is integer. A RETCOD of 0 signifies that the request has processed normally. If RETCOD is non-zero, FOCUS will terminate the report and display:

**(FOC350) ERROR WRITING OUTPUT FILE:**

The error message will include the non-zero value of RETCOD.

- ACVT is a one-word integer; reserved.

In MVS, the subroutine must be allocated to ddname FOCLIB. Compile and link the subroutine as a separate module with AMODE=31, RMODE=24.

In CMS, the program should be compiled, and the TEXT deck available at run time.



## **Example**

### **Sample User-Coded Program That Creates a HOLD File**

This simple COBOL program shows the use of these parameters. It would execute when a report request includes the phrase ON TABLE HOLD VIA EXAMPLE, or when HOLD VIA EXAMPLE is issued from Hot Screen or after a report is displayed:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
INSTALLATION. IBI.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.

INPUT-OUTPUT SECTION.
FILE-CONTROL.

DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.

LINKAGE SECTION.

01 RECNO PIC S9(9) COMP.
01 LEN PIC S9(9) COMP.
01 DDNAME PIC X(8).
01 REC PIC X(4096).
01 RETCOD PIC S9(9) COMP.
01 ACVT PIC S9(9) COMP.

PROCEDURE DIVISION USING RECNO, LEN, DDNAME, REC, RETCOD, ACVT.

PERFORM SHOWPARMS.
GOBACK.

SHOWPARMS.
 DISPLAY " "
 DISPLAY " EXAMPLE COBOL DISPLAY: "
 DISPLAY " RECORD NUMBER " RECNO.
 DISPLAY " LENGTH OF RECORD IS " LEN.
 DISPLAY " DDNAME IS " DDNAME.
 DISPLAY " RECORD IS " REC.
 DISPLAY " RETURN CODE IS " RETCOD.
 DISPLAY " ACVT IS " ACVT.
 MOVE SPACES TO REC.
```

---

# APPENDIX E

## Character Charts

**Topics:**

- Letters
- Numbers
- Punctuation
- Symbols
- Accent Marks and Accented Letters

This appendix contains charts listing the printable EBCDIC character set. EBCDIC is a code that enables IBM S/390 computers to store 256 characters as integers from 0 to 255. The character set in this appendix is based upon the display of an IBM 3270 terminal with the Hot Screen facility deactivated (SET SCREEN=OFF). Use of a 3270 emulator may give different results depending upon the code page used. The printable character sets are organized by topic:

- Letters: uppercase and lowercase.
- Numbers: integer.
- Punctuation: space/blank, common punctuation, brackets, and braces.
- Symbols: common, financial, mathematical, and rare.
- International: accent marks and accented letters.

In the charts, the printable characters appear on the left; their equivalent integers in decimal on the right.

# Letters

| Uppercase Letters | Lowercase Letters | Uppercase Letters | Lowercase Letters |
|-------------------|-------------------|-------------------|-------------------|
| A 193             | a 129             | N 213             | n 149             |
| B 194             | b 130             | O 214             | o 150             |
| C 195             | c 131             | P 215             | p 151             |
| D 196             | d 132             | Q 216             | q 152             |
| E 197             | e 133             | R 217             | r 153             |
| F 198             | f 134             | S 226             | s 162             |
| G 199             | g 135             | T 227             | t 163             |
| H 200             | h 136             | U 228             | u 164             |
| I 201             | i 137             | V 229             | v 165             |
| J 209             | j 145             | W 230             | w 166             |
| K 210             | k 146             | X 231             | x 167             |
| L 211             | l 147             | Y 232             | y 168             |
| M 212             | m 148             | Z 233             | z 169             |

# Numbers

| Numbers | Numbers |
|---------|---------|
| 0 240   | 5 245   |
| 1 241   | 6 246   |
| 2 242   | 7 247   |
| 3 243   | 8 248   |
| 4 244   | 9 249   |

# Punctuation

| Punctuation | Punctuation |
|-------------|-------------|
| space 64    | ' 125       |
| . 75        | “ 127       |
| ! 90        | * 92        |
| ? 111       | / 97        |
| , 107       | ( 77        |
| ; 94        | ) 93        |
| : 122       | [ 65        |
| - 96        | ] 66        |
| _ 109       | { 192       |
|             | } 208       |

# Symbols

| Commonly Used Symbols | Financial Symbols | Mathematical Symbols | Rarely Used Symbols |
|-----------------------|-------------------|----------------------|---------------------|
| @ 124                 | \$ 91             | + 78                 | ⌘ 70                |
| # 123                 | ¢ 74              | - 96                 | ß 71                |
| % 108                 | £ 67              | * 92                 | § 72                |
| & 80                  | ¥ 68              | / 97                 | ‡ 106               |
| * 92                  | € 80              | = 126                | \ 224               |
| ° 81                  |                   | @ 124                | Æ 250               |
|                       |                   | ¬ 95                 | æ 225               |
|                       |                   | 79                   | Ø 251               |
|                       |                   | > 110                | ø 234               |
|                       |                   | < 76                 | ℙ <sub>↑</sub> 69   |

# Accent Marks and Accented Letters

| Accent Marks | Accent Marks |
|--------------|--------------|
| ˘ 73         | ˘ 85         |
| ˘ 82         | ˘ 86         |
| ˆ 83         | ˘ 121        |
| ¨ 84         | ˜ 161        |

| Accented Letters | Accented Letters | Accented Letters | Accented Letters | Accented Letters |
|------------------|------------------|------------------|------------------|------------------|
| Á 203            | À 158            | Â 188            | Ä 183            | Ã 172            |
| á 143            | à 103            | â 138            | ä 117            | ã 100            |
| É 204            | È 159            | Ê 189            | Ë 184            | Ñ 220            |
| é 144            | è 104            | ê 139            | ë 118            | ñ 157            |
| Í 205            | Ì 160            | Î 190            | Ï 185            | Õ 173            |
| í 154            | ì 89             | î 140            | ï 119            | õ 101            |
| Ó 218            | Ò 170            | Ô 191            | Ö 186            | Å 252            |
| ó 155            | ò 98             | ô 141            | ö 120            | å 235            |
| Ú 219            | Ù 171            | Û 202            | Ü 187            | Ç 253            |
| ú 156            | ù 99             | û 142            | ü 128            | ç 236            |
|                  |                  |                  | ÿ 102            |                  |

# Index

## Symbols

- subtraction operator, 8-4 to 8-5
- \$ masking character, 5-20 to 5-22
- \$\* masking character, 5-20, 5-22 to 5-23
- % masking character, 5-20, 5-22
- \* multiplication operator, 8-4 to 8-5
- \*\* exponentiation operator, 8-4 to 8-5
- / division operator, 8-4 to 8-5
- ? FILTER command, 5-32, 5-36
- ? JOIN command, 13-33
- ? STAT command, 4-34
- ? STYLE command, 10-17
- ?F command, 1-16
- ?FF command, 1-17
- \_ masking character, 5-20
- + addition operator, 8-4 to 8-5

## A

- accent marks, E-4
- ACCEPT attribute, 11-19 to 11-20
- Access Files, A-27
  - VIDEOTR2, A-27
- accessing help, 1-4
- ACROSS COLUMNS AND phrase, 4-12, 4-14 to 4-15
- ACROSS phrase, 1-7, 2-3, 2-19 to 2-20, 4-1, 4-5 to 4-7, 10-34
  - COMPUTE command and, 6-15
  - GRAPH, 18-14 to 18-15
- ACROSSCOLUMN, 10-34
  - WHEN, 10-51

- ACROSSTITLE, 10-25
- ACROSS-TOTAL phrase, 4-7 to 4-8
- ACROSSVALUE, 10-25
- ADD command, 2-9
- adding columns in financial reports, 16-27
- adding page breaks, 16-30
- adding underlines to columns, 16-30
- adding virtual fields, 6-7 to 6-8
- addition operator, 8-4 to 8-5
- aggregate values, 5-10
  - selecting records and, 5-10
- aggregation, 4-35, 9-50
  - external sorting and, 4-35 to 4-37
- aliases, 1-13 to 1-14, 19-10
  - displaying, 1-17
  - SQL Translator and, 19-10
- ALL command, 12-17
  - missing values and, 12-17 to 12-18
- ALL parameter, 5-8, 13-6
  - JOIN command and, 13-4, 13-6
  - MULTIPATH parameter and, 5-8 to 5-9
- ALL prefix, 12-16
  - missing values and, 12-16
- ALLOCATE command, 11-2
- ALPHA format, 11-27
- alphanumeric fields, 5-19
  - testing character strings, 5-20 to 5-25
- alternate file views, 6-4, 15-2 to 15-3
- alternate indexes, 5-39
  - selecting records and, 5-39
- alternate views, 6-4
- AND operator, 5-12, 8-14

APT (Automatic Passthru), 19-6  
arithmetic expressions, 8-4  
    creating, 8-4  
arithmetic operators, 8-4 to 8-5  
AS phrase, 9-15 to 9-16, 10-25, 11-3, 11-15  
    in extract files, 3-17  
ascending sort order, 4-10 to 4-11  
AS NAMES command, 11-14 to 11-17  
ASQ prefix operator, 2-17  
assigning row titles, 16-7  
AUTOINDEX parameter, 15-4 to 15-5  
Automatic Passthru (APT), 19-6  
AUTOPATH parameter, 15-4  
AUTOTABLEF parameter, 2-21, 2-23  
AUTOTICK parameter, 18-40, 18-42 to 18-43,  
    18-60  
AVE prefix operator, 2-17

## **B**

BACK command in Hot Screen, 3-7  
bar charts, 18-29  
BAR command, 16-30  
BARNUMB parameter, 18-31, 18-45, 18-60  
BARSPACE parameter, 18-29, 18-31, 18-60  
BARWIDTH parameter, 18-29, 18-31, 18-60  
base dates, 8-8 to 8-9  
BINARY format, 11-27  
BINS parameter, 4-34  
blank lines, 9-8  
    inserting, 9-8 to 9-9  
blanks, 12-11  
    testing for, 12-11  
Boolean expressions, 8-14 to 8-15

Boolean operators, 8-14  
BOTTOM command in Hot Screen, 3-7  
BOTTOMMARGIN parameter, 10-14  
BSTACK parameter, 18-29, 18-60  
BY field in Hot Screen, 3-12  
BY phrase, 1-7, 2-3, 2-10 to 2-11, 2-19, 4-1, 4-3 to  
    4-4, 10-26, 16-10  
    GRAPH, 18-14 to 18-15  
    WITHIN phrase and, 2-29  
BY ROWS OVER phrase, 4-12 to 4-14  
BY TOTAL phrase, 4-25 to 4-26  
BYPANEL parameter, 3-12  
BYSCROLL parameter, 3-9  
byte precision, 2-13  
    expanding, 2-13 to 2-14

## **C**

calculated values, 6-2, 6-11 to 6-13, 7-12  
    in column and row totals, 7-2  
    screening on, 6-16  
    sorting by, 4-25 to 4-27, 6-15  
    subtotals and, 7-12, 7-14 to 7-15  
calculating column and row totals, 7-2 to 7-3  
calculating dates, 8-9  
calculating forecast values, 6-17  
calculating trend values, 6-17  
calculations, 16-22  
    subroutines and, 16-22 to 16-23  
calculations for DEFINE, 6-10  
calculations on field values, 2-15  
    average (AVE), 2-17  
    average sum of squares (ASQ), 2-17 to 2-18  
    column percent (PCT), 2-19  
    counting, 2-27  
    direct percent of counts (PCT.CNT), 2-21  
    maximum value (MAX), 2-18  
    minimum value (MIN), 2-18

- calculations on field values (*continued*)
  - row percent (RPCT), 2-19 to 2-20
  - sum numeric (SUM), 2-27
  - total (TOT), 2-27
  - WITHIN phrase and, 2-29 to 2-30
- CAR data source, A-11
- Cartesian product, 14-25 to 14-26, 19-13
  - SQL Translator and, 19-13
- CA-TELLAGRAF Interface, 18-53
- CDN (Continental Decimal Notation), 19-13
  - SQL Translator and, 19-13
- cells, 16-20
  - identifying, 16-20 to 16-21
- character expressions, 8-3, 8-12
- character strings, 3-10, 8-13
  - concatenating, 8-13
  - locating, 3-10
  - testing, 5-19 to 5-25
- characters, E-1
  - accent marks, E-4
  - accented letters, E-4
  - letters, E-2
  - numbers, E-2
  - printable equivalents, E-1
  - punctuation, E-3
  - symbols, E-3
- CHECK FILE command, 13-32
  - join structures and, 13-31 to 13-32
- CHECK STYLE, 10-19
- class intervals, 18-42
- clearing conditional join structures, 13-34 to 13-35
- clearing join structures, 13-34
- clearing virtual fields, 6-4, 6-6
- CNT prefix operator, 2-27
- COLOR, 10-19
- COLUMN, 10-34
  - column addresses, 16-18 to 16-19
  - column numbers, 16-16
  - column position, 6-14
  - column spacing, 9-25
  - column titles, 9-15
    - creating, 9-15 to 9-16
    - customizing, 9-17
    - justifying, 9-26 to 9-27
  - column totals, 1-10, 4-7 to 4-8, 7-2 to 7-3
    - calculated values and, 7-2, 7-6
    - calculating, 7-2 to 7-4
    - renaming, 7-2, 7-5
  - column values, 16-20
- columns, 1-7
  - adding, 16-27
  - calculating percent, 2-19
  - calculating totals, 7-2 to 7-4
  - calculating values, 6-11 to 6-13
  - compressing, 9-22 to 9-24
  - formatting, 1-19 to 1-20, 1-24
  - identifying, 16-16 to 16-20
  - positioning, 9-18 to 9-21
  - ranking, 4-27
  - sorting by, 4-5 to 4-7
  - summing and sorting, 4-25 to 4-26
- COLUMNS parameter, 3-14
- COLUMN-TOTAL phrase, 7-2 to 7-4
- COM format, 11-29
- COMASTER data source, A-21
- combining records, 16-8 to 16-9
- COMMA format, 11-28
- comma-delimited files, 13-3
- commands, 3-10
  - canceling in Hot Screen, 3-10
  - FOLD-LINE, 9-22
  - MULTILINES, 9-36 to 9-37
  - NOPRINT, 9-11
  - NOSPLIT, 9-5 to 9-6
  - OVER, 9-23
  - PAGE-BREAK, 9-2
  - REPAGE, 9-2



- commands (*continued*)
  - repeating in Hot Screen, 3-10
  - SKIP-LINE, 9-8
  - SUBFOOT, 9-36 to 9-37
  - SUP-PRINT, 9-11
  - UNDER-LINE, 9-10
- common high-order sort fields, 14-9, 14-11, 14-22, 14-24
- comparing characters with a mask, 5-20 to 5-23
- compiling calculations, 6-10
- complex expressions, 8-1
- compound expressions, 5-12
- COMPUTE command, 2-10, 6-2, 6-11 to 6-13, 7-16
  - ACROSS phrase and, 6-15
  - expressions and, 8-2
  - referencing fields, 6-14
  - screening and, 6-16
  - subroutines and, 6-32
- COMT format, 11-30
- concatenating character strings, 8-13
- concatenating data sources, 14-1, 14-14 to 14-16, 14-19
  - field names and, 14-17 to 14-18
- concatenation, 18-17
  - MORE phrase, 18-17
  - universal, 18-17
- concatenation operator, 8-13
- conditional expressions, 8-3, 8-16 to 8-18
- conditional formatting, 9-46 to 9-50
- conditional join structures, 13-2 to 13-3, 13-12, 13-23 to 13-24, 13-31
  - clearing, 13-34 to 13-35
- conditional operators, 5-13, 5-16 to 5-17
- conditional styling, 10-49
- connected point plot graphs, 18-22 to 18-23
- CONTAINS operator, 5-19 to 5-20
- contiguous columns, 16-17
- Continental Decimal Notation (CDN), 19-13
  - SQL Translator and, 19-13
- conversions, E-1
  - code page, E-1
  - printable equivalents, E-1
- COUNT \* command, 2-12
- COUNT command, 2-10 to 2-13, 4-2
  - unique segments, 2-12
- count of occurrences, 2-27
- COUNTWIDTH command, 2-10, 2-13
- COURSES data source, A-16
- CREATE TABLE command, 19-11
- CREATE VIEW command, 19-12
- creating financial reports, 16-3, 16-5 to 16-6, 16-8 to 16-11, 16-24
  - adding columns, 16-27
  - external files and, 16-10
  - formatting, 16-30
  - from multiple records, 16-8
  - identifying a range of values, 16-9
  - identifying cells, 16-20 to 16-21
  - identifying columns, 16-16 to 16-20
  - identifying rows, 16-13 to 16-15
  - inserting text, 16-25 to 16-26
  - inter-row calculations and, 16-12 to 16-13
  - masking and, 16-9
  - recursive models and, 16-28 to 16-29
  - repeating rows, 16-15
  - subroutines and, 16-22 to 16-23
- creating HOLD files, D-1
  - from financial reports, 16-35 to 16-36
- creating reports, 1-1 to 1-3
- creating rows, 16-5 to 16-6
  - from multiple records, 16-8
- creating temporary fields, 1-12, 6-1, 6-4
  - with COMPUTE, 6-2, 6-11 to 6-13
  - with DEFINE, 6-2 to 6-4, 6-6 to 6-7
  - with DEFINE and ADD, 6-7 to 6-8
  - with DEFINE FUNCTION, 6-33
- creating virtual fields, 6-1, 6-4, 6-6 to 6-7, 6-9

cross-century dates, 8-10

cross-referenced fields, 13-15  
  data formats for, 13-21

cross-referenced files, 13-4, 13-12

custom reports, 10-2  
  free-form reports, 17-6

customizing reports, 1-17 to 1-18, 9-1, 9-22, 9-27 to 9-28  
  with SET parameters, 9-27

## D

DATA, 10-24

data descriptions, 1-2

data extraction, 11-3, 11-7

data fields, 5-20  
  testing, 5-20 to 5-21

data formats for join structures, 13-21

data retrieval, 5-5 to 5-6, 15-1 to 15-6, 16-34  
  TABLEF command and, 15-6

data source types, 1-2

data sources, 1-4  
  concatenating, 14-14 to 14-16  
  exporting from, 11-3, 11-7  
  joining, 13-1, 13-3 to 13-4, 13-12, 13-22  
  merging, 14-1 to 14-7, 14-9, 14-11 to 14-12, 14-19 to 14-20, 14-22, 14-24  
  missing values, 12-1  
  multi-path, 2-6  
  rotating, 15-2

data structures, 13-1, 15-3

data type conversions, 13-22

data types, 13-22  
  converting for join structures, 13-22

date components, 8-11  
  extracting, 8-11

date constants, 8-7, 8-10

date expressions, 8-3, 8-7, 8-10  
  combining fields, 8-11

date fields, 8-7  
  extracting components from, 8-11

date formats, 8-8 to 8-9, 19-15 to 19-16  
  calculations in, 8-10  
  SQL Translator and, 19-15 to 19-16

date values, 8-8  
  format for, 8-8 to 8-9

dates, 8-9  
  extracting components, 8-11  
  in graphs, 18-47  
  performing calculations on, 8-9

date-time expressions, 8-7

date-time values, 8-7, 19-17 to 19-19  
  SQL Translator and, 19-17 to 19-19

DB2 format, 11-30

DBAFILE attribute, 13-4  
  join structures and, 13-4

deferred graphics output, 18-51

DEFINE attribute, 8-2  
  expressions and, 8-2  
  GRAPH, 18-20

DEFINE command, 6-2 to 6-4, 6-7  
  expressions and, 8-2  
  join structures and, 6-11, 13-17, 13-19 to 13-20, 13-26 to 13-27, 13-29  
  missing values and, 12-4 to 12-8  
  speed of calculations for, 6-10  
  subroutines and, 6-32

DEFINE FILE RETURN command, 6-10 to 6-11, 13-29

DEFINE FILE SAVE command, 6-10 to 6-11, 13-29

DEFINE FUNCTION command, 6-33

DEFINE functions, 6-33  
    calling, 6-33  
    deleting, 6-33, 6-36  
    limitations, 6-33 to 6-34  
    querying, 6-33, 6-35

defining custom groups, 4-17 to 4-18

DELETE command, 19-22

deleting virtual fields, 6-4, 6-6

descending sort order, 4-10, 4-12

DEVICE parameter, 18-10, 18-60

DFSORT sorting product, 4-33  
    CMS requirements, 4-33  
    MVS requirements, 4-33

Dialect Translation, 19-2

Dialogue Manager, 8-2, 16-2

DIF format, 11-31

direct percent, 2-21

display commands, 1-6 to 1-7, 2-1 to 2-2, 4-2  
    ADD, 2-9  
    COMPUTE command and, 6-15  
    COUNT, 2-10, 4-2  
    COUNT \*, 2-12  
    LIST, 2-1, 2-3, 4-2  
    LIST \*, 2-5  
    MATCH FILE command and, 14-12  
    multiple, 4-31, 4-32  
    PRINT, 2-1, 2-3, 4-2  
    PRINT \*, 2-5  
    SUM, 2-1, 2-9 to 2-10, 4-2  
    WRITE, 2-9

display fields, 1-20, 2-15  
    limitations, 1-13  
    prefix operators and, 2-15

displaying data, 1-6 to 1-7

displaying field names, 1-16 to 1-17

displaying grand totals, 7-6 to 7-7

displaying graphs, 18-7

displaying join structures, 13-31 to 13-33

displaying reports, 3-1 to 3-2, 3-11, 3-17, 9-52

displaying subtotals, 7-6 to 7-12

distinct prefix operators, 2-21 to 2-22  
    restrictions, 2-21, 2-23

division operator, 8-4 to 8-5

DOWN command in Hot Screen, 3-7

DROP VIEW command, 19-12  
    SQL Translator and, 19-13

DST prefix operator, 2-21 to 2-22  
    restrictions, 2-21, 2-23

dynamic reformatting, 1-21 to 1-22

## E

EBCDIC, E-1  
    code page, E-1  
    printable characters, E-1

EDUCFILE data source, A-7

embedded quotation marks, 8-12 to 8-13

EMPDATA data source, A-17

EMPLOYEE data source, A-3

empty reports, 3-6, 9-52

EMPTYREPORT parameter, 3-6, 9-52

EMR. *See* FML (Financial Modeling Language)

END command, 1-5, 18-12  
    in GRAPH request, 18-12

ending a report request, 1-5

EQ operator, 5-19, 8-14

equijoins, 13-2 to 13-3, 13-13

error files, B-1  
    CMS, B-2  
    MVS, B-2

error messages, 1-4, B-1  
    displaying, 4-39, B-3

escape characters, 5-24 to 5-25

- estimating number of records, 4-39
  - ESTRECORDS parameter, 4-38 to 4-39
  - EXCEL format, 11-31
  - EXCEPT operator, 19-15
  - EXCLUDES operator, 5-25 to 5-26
  - existing data, 5-19
    - testing for, 5-19, 12-10
  - EXPAVE method, 6-17 to 6-18, 6-21, 6-25 to 6-26
  - EXPERSON data source, A-18
  - exponential moving average, 6-25 to 6-26
  - exponentiation operator, 8-4 to 8-5
  - expression types, 8-3
    - Boolean, 8-14
    - conditional, 8-16
    - date-time, 8-7
    - logical, 8-14
    - relational, 8-14 to 8-15
  - expressions, 8-1 to 8-2, 19-15
    - combining, 5-12
    - COMPUTE command and, 8-2
    - DEFINE attribute and, 8-2
    - DEFINE command and, 8-2
    - field formats and, 8-3 to 8-4
    - IF phrase and, 8-2
    - RECAP command and, 8-2
    - SQL Translator and, 19-15
    - WHEN phrase and, 8-2
    - WHERE phrase and, 8-2
  - EXTAGGR parameter, 4-35
  - external sorting, 4-33 to 4-34
    - aggregation and, 4-35 to 4-37
    - displaying error messages, 4-39
    - HOLD files and, 4-37 to 4-38
    - National Language Support (NLS) and, 4-33
    - requirements, 4-33
    - verifying, 4-34
  - EXTHOLD parameter, 4-37 to 4-38
  - extract files, 11-2
    - displaying, 18-53
    - missing values and, 12-12
    - naming, 11-2
  - EXTRACT function, 19-19
  - EXTSORT parameter, 4-33 to 4-34
- ## F
- field formats, 3-11, 8-8
    - expressions and, 8-3 to 8-4
    - internal storage and, 8-9
    - redefining, 3-11
  - field names, 1-13, 19-14
    - aliases, 1-13 to 1-14
    - controlling, 11-14 to 11-17
    - displaying, 1-16 to 1-17
    - long, 1-14 to 1-15
    - qualified, 1-13 to 1-15
    - SQL Translator and, 19-14
    - truncated, 1-13 to 1-14
  - field padding, 11-42 to 11-43
  - field values, 2-1, 9-41
    - calculating, 6-11 to 6-13
    - computing the average, 2-17
    - counting, 2-10
    - displaying, 2-1, 2-3 to 2-5
    - embedding, 9-41
    - summing, 4-2
    - WITHIN phrase and, 2-29
  - field-based reformatting, 1-21 to 1-23
  - FIELDNAME command, 1-15
  - FIELDNAME SET parameter, 6-4
  - fields, 1-13
    - formatting, 8-10
    - in free-form reports, 17-7
    - in report requests, 1-13 to 1-14
    - joining, 13-22
    - repeating in join structures, 13-7
    - suppressing display, 9-11 to 9-13
    - temporary, 1-12, 6-1
  - FILE command, 1-4

- FILEDEF command, 11-2
- FILTER parameter, 5-32, 5-35
- FILTER query command, 5-32, 5-36
- filtering, 5-2
- filters, 5-32, 5-35 to 5-36
  - defining, 5-32 to 5-34
  - join structures and, 5-38
  - virtual fields and, 5-32, 5-34
- FINANCE data source, A-14
- Financial Modeling Language (FML), 16-1 to 16-3
  - Dialog Manager and, 16-2
  - saving report results, 16-32
- Financial Reporting Language (FRL). *See* Financial Modeling Language (FML)
- financial reports, 16-1 to 16-2
  - creating, 16-3, 16-5 to 16-6, 16-8 to 16-29
  - formatting, 16-30
  - HOLD files and, 16-35 to 16-36
  - posting data, 16-33
  - saving intermediate results, 16-32
  - suppressing rows, 16-31 to 16-32
- financial symbols, E-3
  - printable characters, E-3
- fixed axis scales (fixed limits), 18-41, 18-43, 18-50
- FIXRETRIEVE parameter, 11-21 to 11-22
- FML (Financial Modeling Language), 16-1 to 16-3
  - Dialog Manager and, 16-2
  - saving report results, 16-32
- FOCFIELDNAME amper variable, 1-15
- FOCSTYLE files, 10-7, 10-17
- FOCUS data sources, 5-2 to 5-3, 11-9
  - creating, 11-9, 11-12 to 11-13
  - selecting records and, 5-3
- FOCUS file structure, 11-9, 11-11
- FOCUS format, 11-32
- FOLD-LINE command, 9-16, 9-22 to 9-23
- FONT, 10-19
- footers, 18-18
  - embedding field values, 18-20
  - in free-form reports, 17-6
- footings, 9-29, 10-28
  - creating, 18-18
  - inserting data in, 9-41, 9-44
  - TABLE, 10-28
- FOR command, C-5
- FOR phrase, 2-21, 2-23, 4-12 to 4-13, 4-16 to 4-18, 16-3, 16-10
- FORECAST, 6-17 to 6-18, 6-20, 6-29 to 6-32
- forecast values, 6-17
- formats, 3-17
  - extracting files, 3-17
- formatted graphs, 18-51
  - saving, 18-51
- formatting financial reports, 16-30
  - WebFOCUS StyleSheets and, 16-30
- formatting graphs, 18-7
- formatting report columns, 1-19 to 1-20, 1-24
- formatting reports, 9-1 to 9-2, 9-8, 9-11, 9-17 to 9-18, 9-22 to 9-23, 9-25 to 9-28, 9-30, 9-33, 9-35 to 9-36, 9-39, 9-46, 19-6
  - SQL Translator and, 19-6
- FORWARD command in Hot Screen, 3-7
- free-form reports, 9-45, 17-1 to 17-2
  - designing, 17-6
  - example, 17-2
  - fields, 17-7
  - footers, 17-6
  - formatting, 17-8
  - graphics, 17-7
  - headers, 17-6
  - prefix operators, 17-7
  - selecting records, 17-8
  - sorting, 17-8
  - text, 17-6
- FROM ... TO operator, 5-16 to 5-17
- FST prefix operator, 2-24 to 2-25

function keys in Hot Screen, 3-6, 3-14, 3-17

functions, 8-1

FUSION format, 11-32

## G

GCOLOR parameter, 18-60

GDDM graphics, 18-55

GE operator, 5-17 to 5-18, 8-14

GGDEMOG data source, A-29

GGORDER data source, A-30

GGPRODS data source, A-31

GGSALES data source, A-32

GGSTORES data source, A-33

GMISSING parameter, 18-48, 18-60

GMISSVAL parameter, 18-48, 18-60

Gotham Grinds data sources, A-29

GGDEMOG, A-29

GGORDER, A-30

GGPRODS, A-31

GGSALES, A-32

GGSTORES, A-33

GPPROMPT parameter, 18-7, 18-60

grand totals, 1-10, 7-6

displaying, 7-6 to 7-7

suppressing, 7-19 to 7-20

GRANDTOTAL, 10-26

GRAPH command, 18-2, 18-12, 18-57

graph forms, 18-21

bar charts, 18-29

connected point plots, 18-22, 18-23

histograms, 18-26

pie charts, 18-33

scatter diagrams, 18-35

graph headings, 18-18

GRAPH parameters, 18-60

AUTOTICK, 18-40, 18-60

BARNUMB, 18-31, 18-45, 18-60

BARSPACE, 18-29, 18-31, 18-60

BARWIDTH, 18-29, 18-31, 18-60

BSTACK, 18-29, 18-60

DEVICE, 18-10, 18-60

GCOLOR, 18-60

GMISSING, 18-48, 18-60

GMISSVAL, 18-48, 18-60

GPPROMPT, 18-7, 18-60

GRAPH, 18-38

GRIBBON, 18-60

GRID, 18-35, 18-45, 18-60

GTREND, 18-35, 18-46, 18-60

HAUTO, 18-41, 18-60

HAXIS, 18-40, 18-60

HCLASS, 18-42, 18-60

HISTOGRAM, 18-28, 18-60

HMAX, 18-41, 18-60

HMIN, 18-41, 18-60

HSTACK, 18-28, 18-60

HTICK, 18-42, 18-60

PAUSE, 18-55, 18-60

PIE, 18-33, 18-60

PLOT, 18-60

PRINT, 18-10, 18-60

TERMINAL, 18-60

VAUTO, 18-43, 18-60

VAXIS, 18-43, 18-60

VCLASS, 18-44, 18-60

VGRID, 18-37, 18-45, 18-60

VMAX, 18-43, 18-60

VMIN, 18-43, 18-60

VTICK, 18-44, 18-60

VZERO, 18-48, 18-60

GRAPH requests, 18-12, 18-18

ACROSS phrase, 18-14 to 18-15

BY phrase, 18-14 to 18-15

concatenating files, 18-17

END command, 18-12, 18-59

IF phrase, 18-16

pie charts, 18-33

QUIT command, 18-59

- graph types, 18-1 to 18-2, 18-21
  - bar charts, 18-29
  - connected point plots, 18-22 to 18-23
  - histograms, 18-26
  - pie charts, 18-33
  - scatter diagrams, 18-35

GRAPH vs. TABLE, 18-2, 18-12

graphic devices, 18-10, 18-60

graphics

- in free-form reports, 17-7

graphs, 18-1 to 18-2

- adding footings, 18-18
- adding grids, 18-45
- adjusting parameter settings, 18-38, 18-60
- annotating, 18-18
- class and tick intervals, 18-42
- creating from unlike data sources, 18-17
- deferred output, 18-52
- displaying, 18-7
- displaying stored graphs, 18-52
- fixed axis scales, 18-50
- formatting, 18-7, 18-60
- horizontal axis features, 18-14 to 18-15, 18-40, 18-60
- missing data, 18-48, 18-60
- naming subjects, 18-14
- parameter settings, 18-60
- plotting dates, 18-47
- printer/plotter selection, 18-10, 18-60
- prompting for values, 18-7
- redisplaying with REPLOT, 18-2
- saving, 18-51
- saving formatted graphs, 18-54
- selecting records, 18-16
- stacking bars with OVER, 18-26
- verb phrases, 18-14
- vertical axis features, 18-43

GRIBBON parameter, 18-60

GRID parameter, 18-7, 18-35, 18-45, 18-60

group fields, 13-16

- join structures and, 13-16

group key values, 5-26

grouping numeric data, 4-16 to 4-18

- into tiles, 4-19 to 4-23

GT operator, 5-17 to 5-18, 8-14

GTREND parameter, 18-35, 18-46, 18-60

## H

HAUTO parameter, 18-41, 18-60

HAXIS parameter, 18-40, 18-60

HCLASS parameter, 18-42, 18-60

headers and footers, 18-18

- embedding field values, 18-20

headers in free-form reports, 17-6

headings, 9-29, 10-28

- creating, 9-33, 18-18
- inserting data in, 9-41 to 9-42, 9-44

headings and footings, 10-28

HELP from Hot Screen, 3-6

help reports, 1-4

Hewlett-Packard plotters, 18-56

hiding rows, 16-31 to 16-32

hiding sort field values, 4-29

hierarchy in StyleSheets, 10-43

high-order sort fields, 14-9, 14-11, 14-22, 14-24

high-resolution graphic devices, 18-10 to 18-11

- Hewlett-Packard plotters, 18-56, 18-60
- IBM devices and GDDM, 18-55, 18-60
- Tektronics terminals, 18-57, 18-60

HISTOGRAM parameter, 18-28, 18-60

histograms, 18-26

- HAXIS, 18-40, 18-60
- VAXIS, 18-43, 18-60

HMAX parameter, 18-41, 18-60

HMIN parameter, 18-41, 18-60

HOLD AT CLIENT command, 11-3, 11-6

- 
- HOLD command, 11-2 to 11-3
  - HOLD files, 11-2 to 11-3, 11-10, 14-2
    - creating, 11-3 to 11-4, 11-6 to 11-7, 11-9 to 11-12, 11-42 to 11-44, D-1
    - external sorting and, 4-37 to 4-38
    - financial reports and, 16-35 to 16-36
    - formatting, 11-3
    - keyed retrieval and, 11-21 to 11-22
    - merge phrases and, 14-5 to 14-7
    - missing values and, 12-12
    - querying, 11-3, 11-8
    - suppressing field padding, 11-42 to 11-43
    - text fields and, 11-26, 11-40 to 11-41
  - HOLD formats, 11-26, 11-38
    - ALPHA, 11-27
    - BINARY, 11-27
    - COM, 11-29
    - COMMA, 11-28
    - COMT, 11-30
    - DB2, 11-30
    - DIF, 11-31
    - EXCEL, 11-31
    - FOCUS, 11-32
    - FUSION, 11-32
    - HTML, 11-32
    - HTMTABLE, 11-33
    - INGRES, 11-33
    - INTERNAL, 11-33, 11-42 to 11-43
    - LOTUS, 11-34
    - PDF, 11-34
    - PS, 11-34
    - Red Brick, 11-35
    - SQL, 11-35
    - SQLDBC, 11-35
    - SQLINF, 11-36
    - SQLMSS, 11-36
    - SQLODBC, 11-36
    - SQLORA, 11-37
    - SQLSYB, 11-37
    - SYLK, 11-37
    - WP, 11-38
  - HOLD Master Files, 11-3, 11-8
    - controlling attributes, 11-14, 11-18 to 11-20
    - displaying, 11-3, 11-8
    - field names and, 11-14 to 11-17
  - HOLDATTR command, 11-14, 11-19 to 11-20
  - HOLDLIST command, 11-14, 11-18 to 11-19
  - horizontal axis features, 18-40
    - class and tick intervals, 18-42
    - grids, 18-45
    - scale, 18-43
    - sorting graph subjects, 18-14 to 18-15
    - width, 18-40
  - horizontal bar charts, 18-29
  - host fields, 13-21
    - data formats for, 13-21
  - host files, 13-4, 13-12
  - Hot Screen, 3-8, 3-17
    - activating, 3-3
    - canceling commands, 3-10
    - displaying BY fields with panels, 3-12
    - function keys, 3-6, 3-14, 3-17
    - help information, 3-6
    - locating character strings, 3-10
    - panel, 3-15
    - previewing reports, 3-12
    - printing, 3-1
    - redisplaying reports, 3-11
    - reissuing previous command, 3-10
    - repeating commands, 3-10
    - SAVE files, 3-9
    - saving selected data, 3-9
    - scrolling, 3-7 to 3-8, 3-14
    - unusual character display, E-1
  - Hot Screen commands, 3-7
    - BACK, 3-7
    - BOTTOM, 3-7
    - DOWN, 3-7
    - FORW(ARD), 3-7
    - LEFT, 3-8
    - NEXT, 3-7
    - OFFLINE, 3-16
    - OFFLINE CLOSE, 3-16
    - RESET, 3-8
    - RETYPE, 3-11, 3-16
    - RIGHT, 3-8
    - TOP, 3-7
    - UP, 3-7



HSTACK parameter, 18-28, 18-60

HTICK parameter, 18-42, 18-60

HTML format, 11-32

HTMTABLE format, 11-33

## I

ICU (Interactive Chart Utility) Interface, 18-14, 18-54

identifying a range of values, 16-9

identifying cells, 16-20 to 16-21

identifying columns, 16-16, 16-18 to 16-20

identifying contiguous columns, 16-17

identifying rows, 16-13 to 16-15

IF ... THEN ... ELSE expressions, 8-16

IF command, 18-16

IF operator, 5-13, 5-15

IF phrase, 5-2, 5-28 to 5-29  
    expressions and, 8-2  
    selecting records and, 5-30, 5-32

IN phrase, 9-18 to 9-21

INCLUDES operator, 5-25 to 5-26

independent paths, 5-5  
    selection criteria and, 5-5 to 5-6

index optimized retrieval, 19-20

INGRES format, 11-33

IN-GROUPS-OF phrase, 4-16 to 4-17

inheritance in StyleSheets, 10-43

INSERT command, 19-22

INSERT INTO command, 19-11

inserting text in financial reports, 16-25 to 16-26

INTERNAL format, 11-33, 11-42 to 11-43

internal matrixes, 15-7  
    saving, 15-7

international printable characters, E-4

INTERSECT operator, 19-15

irrelevant report data, 12-1 to 12-2

IS NOT operator, 5-20 to 5-21

IS operator, 5-20 to 5-21

ITEM, 10-29

## J

JOBFILE data source, A-6

JOIN CLEAR command, 13-34 to 13-35

JOIN command, 13-12 to 13-13, 13-17, 13-23, 19-8 to 19-10  
    ALL parameter and, 13-4, 13-6  
    recursive structures, 13-7, 13-9  
    SQL Translator and, 19-8 to 19-10  
    supported data sources, 13-3

join structures, 5-38, 13-1, 13-3, 13-12 to 13-13, 13-15, 19-8 to 19-9  
    clearing, 13-34 to 13-35  
    cross-referenced fields, 13-15  
    DBA security and, 13-4  
    DEFINE command and, 6-11, 13-17, 13-19 to 13-20, 13-26 to 13-27, 13-29  
    displaying, 13-31 to 13-33  
    group fields and, 13-16  
    optimizing, 19-20  
    qualified field names and, 19-10  
    WHERE phrase and, 13-31

join types, 13-2  
    conditional, 13-2  
    equijoins, 13-2

JOINOPT parameter, 13-22

justifying column titles, 9-26 to 9-27

## K

KEEPDEFINES parameter, 13-26 to 13-27  
 key fields, 11-9, 11-11  
 keyed retrieval, 11-21  
     HOLD files and, 11-21 to 11-22

## L

LE operator, 5-17 to 5-18, 8-14  
 LEDGER data source, A-13  
 LEFT command in Hot Screen, 3-8  
 LEFTGAP, 10-42  
 LEFTMARGIN parameter, 10-15  
 letters, E-2  
 LIKE operator, 5-20 to 5-21  
 limits for display fields, 1-13  
 LINE, 10-29  
 linear regression, 6-27 to 6-28  
 LIST \* command, 2-5  
 LIST command, 2-1, 2-3, 2-13, 4-2  
 listing join structures, 13-33  
 literals, 19-15  
 load procedures, A-1 to A-2  
 LOCATE command, 3-10  
     TABLE, 3-10  
 locating character strings, 3-10  
 logical expressions, 5-12, 5-19, 8-3, 8-14  
 logical operators, 5-12, 8-14  
     CONTAINS, 5-19 to 5-20  
     OMITS, 5-19 to 5-20  
 long field names, 1-13 to 1-15  
 LOTUS format, 11-34  
 low-resolution graphic devices, 18-10

LST prefix operator, 2-24  
 LT operator, 5-17 to 5-18, 8-14

## M

masked fields, 5-20 to 5-22  
 masking, 16-9  
 masking characters, 5-20, 5-22  
     treating as literals, 5-24 to 5-25  
 masks, 5-20 to 5-21  
 Master Files, 1-2  
     filters and, 5-32  
     HOLD files, 11-3  
     MISSING attribute and, 12-4 to 12-5  
     samples, A-1 to A-2  
 MATCH command, 14-2 to 14-5  
 MATCH FILE command, 14-2 to 14-7, C-4  
     concatenated data sources and, 14-19 to 14-20  
     display commands and, 14-12  
     merge phrases and, 14-6  
 mathematical symbols, E-3  
 matrix reports, 4-9  
     calculating row and column totals, 7-2, 7-4  
     creating, 4-9 to 4-10  
 matrixes, 1-7  
     creating, 1-9  
     internal, 15-7  
     saving, 15-7  
 MAX prefix operator, 2-18  
 medium-resolution graphic devices, 18-10 to 18-11, 18-55  
     Anderson Jacobson, 18-60  
     Gencom, 18-60  
 merge phrases, 14-5, 14-7  
     HOLD files and, 14-5 to 14-6  
     MATCH FILE command and, 14-5 to 14-6  
 merging data sources, 14-1 to 14-7, 14-9, 14-11, 14-19 to 14-20, 14-22, 14-24  
     display commands and, 14-12

MIN prefix operator, 2-18

MISSING attribute, 5-19, 12-3, 12-9 to 12-11

- extract files and, 12-12
- limits, 12-5
- Master Files and, 12-4 to 12-5
- virtual fields and, 12-5

missing values, 5-19, 12-1, 12-3 to 12-4

- ALL command and, 12-17 to 12-18
- ALL prefix and, 12-16
- DEFINE command and, 12-4 to 12-8
- designating, 12-20
- excluding from tests, 12-11
- extract files and, 12-12
- GRAPH requests, 18-48, 18-60
- irrelevant report data, 12-2
- segment instances and, 12-3, 12-13 to 12-15
- temporary fields and, 12-6
- testing for, 12-9, 12-19

MORE phrase, 14-14 to 14-16, 14-19, 18-17

MOVE method, 6-17 to 6-18, 6-21 to 6-22, 6-24

MOVIES data source, A-24

MULTILINES command, 7-8, 7-12, 9-36 to 9-37

multi-path data sources, 2-6, 2-8

- displaying the structure, 2-6
- sort fields and, 4-3, 4-6
- virtual fields and, 6-9

MULTIPATH parameter, 5-5 to 5-6

- ALL parameter and, 5-5, 5-8
- segments and, 5-5, 5-9

multiple records, 16-8

multiple sort fields, 4-4 to 4-5, 4-7

multiple virtual fields, 6-7 to 6-8

multiplication operator, 8-4 to 8-5

multi-segment data sources, 5-25

- selecting records and, 5-25 to 5-26

## N

National Language Support (NLS), 4-33

NE operator, 5-19, 8-14

NEXT command in Hot Screen, 3-7

NLS (National Language Support), 4-33

NODATA character, 12-2 to 12-3, 12-20

- setting, 12-20

non-numeric fields, 2-9 to 2-10

non-recursive models, 16-28

non-unique join structures, 13-2, 13-4, 13-6

NOPAGE command, 9-5

NOPRINT command, 4-29, 9-11 to 9-12, 16-31

NOSPLIT command, 9-5 to 9-6

NOT FROM ... TO operator, 5-16 to 5-17

NOT LIKE operator, 5-20 to 5-21

NOT operator, 8-14

NOTOTAL command, 7-19 to 7-20

numbers, E-2

- printable characters, E-2

numeric constants, 19-15

numeric data types, 13-22

- joins and, 13-22

numeric expressions, 8-3

- creating, 8-4
- operators, 8-4 to 8-5
- order of evaluation and, 8-5 to 8-6

numeric fields, 2-9

- adding values, 2-9

## O

- OBJECT, 10-29
- OFFLINE CLOSE, 3-16
- OFFLINE command, 3-16
  - TABLE, 3-1
- offline printing in Hot Screen, 3-16
- OMITS operator, 5-19 to 5-20
- ON GRAPH command, 18-51
- ON phrase, 9-46 to 9-47
- ON TABLE, C-2
- ONLINE command, 3-1
  - TABLE, 3-1
- ONLINE-FMT parameter, 10-12
- operators, 8-14
  - arithmetic, 8-4 to 8-5
  - Boolean, 8-14
  - logical, 5-12, 5-19, 8-14
  - prefix, 2-15 to 2-16
  - relational, 5-13, 5-15 to 5-19, 8-14
- optimized join structures, 19-20
- OR operator, 5-12, 8-14, 16-8
- order of evaluation, 8-5 to 8-6
  - numeric expressions and, 8-5 to 8-6
- ORIENTATION parameter, 10-14
- output file formats, 11-26
  - ALPHA, 11-27
  - BINARY, 11-27
  - COM, 11-29
  - COMMA, 11-28
  - COMT, 11-30
  - DB2, 11-30
  - DIF, 11-31
  - EXCEL, 11-31
  - FOCUS, 11-32
  - FUSION, 11-32
  - HTML, 11-32
  - HTMTABLE, 11-33
  - INGRES, 11-33
  - output file formats (*continued*)
    - INTERNAL, 11-33
    - LOTUS, 11-34
    - PDF, 11-34
    - PS, 11-34
    - Red Brick, 11-35
    - SQL, 11-35
    - SQLDBC, 11-35
    - SQLINF, 11-36
    - SQLMSS, 11-36
    - SQLODBC, 11-36
    - SQLORA, 11-37
    - SQLSYB, 11-37
    - SYLK, 11-37
    - WP, 11-38
- output files, 11-2
  - creating, 11-3
  - formatting, 11-26
  - missing values and, 12-12
  - naming, 11-2
  - saving, 11-2
  - text fields and, 11-26, 11-40 to 11-41
- OVER command, 9-22 to 9-24
  - GRAPH, 18-26

## P

- padded fields, 11-42 to 11-43
- page breaks, 9-2 to 9-3, 9-5
  - suppressing, 9-5 to 9-6
- page footings, 9-33
  - creating, 9-34
- page headings, 9-30
  - creating, 9-30, 9-32
- page numbers, 9-2, 9-4
  - inserting, 9-4
  - suppressing, 9-5
- PAGE parameter, 9-5
- PAGE-BREAK command, 9-2 to 9-3, 16-30
- PAGENUM parameter, 10-24
- pages, 9-2
  - formatting with StyleSheets, 10-13

- PAGESIZE parameter, 10-16
- panels, 3-15
  - viewing reports, 3-15
- parent instances, 12-17 to 12-18
- PAUSE parameter, 18-55
  - GRAPH, 18-60
- PAYHIST data source, A-20
- PCHOLD command, 11-2 to 11-3, 11-6
- PCHOLD files, 11-3
  - creating, 11-3, 11-6
  - formatting, 11-3, 11-6
- PCHOLD formats, 11-26
  - ALPHA, 11-27
  - BINARY, 11-27
  - DIF, 11-31
  - EXCEL, 11-31
  - HTML, 11-32
  - HTMTABLE, 11-33
  - LOTUS, 11-34
  - PDF, 11-34
  - WP, 11-38
- PCT prefix operator, 2-19
- PCT.CNT prefix operator, 2-21
- PDF format, 11-34
- percentiles, 4-19
- performance, 4-33, 15-1, 19-21
  - improving, 4-33, 4-37, 13-13, 15-1, 19-21
- pie charts, 18-33
- PIE parameter, 18-33, 18-60
- PLOT parameter, 18-60
- POSITION attribute, 10-38
- positional labels, 16-13 to 16-15
- positioning text, 9-39 to 9-40
- POST command, 16-33
- posting data to a file, 16-33
- PostScript (PS) files, 10-7, 10-12
- PostScript format, 11-34
- precision, 2-13
  - expanding, 2-13 to 2-14
- prefix operators, 2-15 to 2-16
  - ASQ, 2-17 to 2-18
  - AVE, 2-17
  - CNT, 2-27
  - display fields and, 2-15
  - distinct, 2-21 to 2-22
  - DST, 2-21 to 2-22
  - FST, 2-24
  - GRAPH, 18-14
  - in free-form reports, 17-7
  - LST, 2-24
  - MAX, 2-18
  - MIN, 2-18
  - PCT, 2-19
  - PCT.CNT, 2-21
  - RPCT, 2-19 to 2-20
  - SUM, 2-27
  - TOT, 2-27 to 2-28
- preserving field names, 11-14
- preserving missing values, 12-12
- PRINT \* command, 2-5
- PRINT command, 2-1, 2-3, 2-5, 4-2
  - GRAPH, 18-10, 18-60
  - merging data sources and, 14-12
  - unique segments, 2-5 to 2-6, 2-8
- printable characters, E-4
- printer/plotter selection for graphs, 18-10, 18-60
- printing StyleSheets, 10-12
- PRINTONLY parameter, 11-18 to 11-19
- PRINTPLUS parameter, 3-4
- procedures, A-1
  - load, A-1 to A-2
- PROD data source, A-10
- PS (PostScript) files, 10-7, 10-12
- punctuation, E-3
  - accent marks, E-4
  - printable characters, E-3

## Q

qualified field names, 1-13 to 1-15, 19-10, 19-14  
     SQL join structures and, 19-10  
     SQL Translator and, 19-14

qualified field values, 2-29, 5-25  
     in parent segments, 5-25  
     WITHIN phrase and, 2-29

QUALTITLES parameter, 9-17

quartiles, 4-19

query commands, 10-17  
     ? STYLE, 10-17  
     ?F, 1-16  
     ?FF, 1-17

QUIT command, 1-5  
     in GRAPH request, 18-12, 18-59

quotation marks, 8-12

quote-delimited strings, 8-12 to 8-13

## R

range of values, 16-9

range tests, 5-16 to 5-18

ranges, 4-16 to 4-18

RANKED BY phrase, 4-27 to 4-28

RANKED BY TOTAL phrase, 4-27

ranking sort field values, 4-24 to 4-25, 4-27 to 4-28

READLIMIT operator, 5-27

RECAP command, 7-16 to 7-19, 10-26, 16-3,  
     16-12 to 16-13  
     expressions and, 8-2

RECOMPUTE command, 7-12 to 7-13, 7-15

RECORDLIMIT operator, 5-27 to 5-28

records, 1-10  
     combining, 16-8 to 16-9  
     comparing, 14-5 to 14-7  
     listing, 2-3 to 2-4  
     retrieving, 2-24, 5-27 to 5-28

records (*continued*)  
     selecting, 1-10, 5-1, 5-5 to 5-6, 5-10 to 5-13,  
         5-15 to 5-16, 5-19 to 5-20, 5-25 to 5-26, 5-28 to  
         5-32  
     selecting in free-form reports, 17-8  
     sorting in free-form reports, 17-8

recursive join structures, 13-7, 13-9 to 13-10, 19-10

recursive models, 16-28 to 16-29

Red Brick format, 11-35

reformatting fields, 1-21 to 1-22, 1-23

REGION data source, A-15

REGRESS method, 6-17 to 6-18, 6-21, 6-27 to 6-28

relational expressions, 5-12, 8-14 to 8-15

relational operators, 5-13, 5-15 to 5-19, 8-14  
     EXCLUDES, 5-25 to 5-26  
     INCLUDES, 5-25 to 5-26  
     IS, 5-20 to 5-21  
     LIKE, 5-20 to 5-21  
     NOT IS, 5-20 to 5-21  
     NOT LIKE, 5-20 to 5-21  
     READLIMIT, 5-27  
     RECORDLIMIT, 5-27 to 5-28  
     testing multi-segment files, 5-25  
     wildcard characters and, 5-20, 5-22

REPAGE command, 9-2

repeating fields, 13-7

repeating rows, 16-15

REPLOT command, 18-2  
     GRAPH, 18-2

REPORT, 10-24

report columns, 4-25  
     formatting, 1-19 to 1-20, 1-24  
     summing and sorting, 4-25 to 4-26

report components, 10-21

report footings, 9-29, 9-33  
     creating, 9-33

report headings, 9-29 to 9-30  
     creating, 9-30 to 9-31

- report panels, 3-14
- report requests, 1-6, 19-1
  - SQL statements and, 19-1
- report types, 4-9
  - free-form, 9-45, 17-1 to 17-2
  - matrix, 4-9
- reports, 1-1 to 1-3, 1-13, 3-1
  - comparing styled and non-styled, 10-8
  - customizing, 1-17 to 1-18, 9-1, 10-2
  - displaying, 1-5, 3-1 to 3-2, 3-11, 3-17, 9-52
  - displaying data, 1-6 to 1-7, 5-1
  - formatting, 7-21, 9-2 to 9-5, 9-8 to 9-11, 9-15, 9-17 to 9-18, 9-22 to 9-23, 9-25 to 9-28, 9-30, 9-33, 9-35 to 9-36, 9-39, 9-46
  - formatting with StyleSheets, 10-1
  - printing, 1-5, 3-17
  - reusing output, 11-1
  - running, 1-5
  - saving, 1-5, 1-24, 11-1 to 11-2
  - scrolling in Hot Screen, 3-7
  - selecting records, 1-10
  - sorting, 1-7 to 1-9, 4-1 to 4-7, 4-10, 4-12, 4-16, 4-27, 4-29
  - syntax summary, C-1
- reserved words, 19-4
- RESET command in Hot Screen, 3-8
- restricting sort field values, 4-24 to 4-25, 4-27 to 4-28
- restructuring data, 15-3
- retrieval limits, 5-27 to 5-28
- retrieval logic, 15-2
- retrieval order, 4-37
  - setting, 4-37
- retrieving data, 16-34
- retrieving records, 5-27 to 5-28, 13-13
- returned fields, 8-10
- RETYPE command, 3-11, 3-16
- reusing report output, 1-24, 11-1
- RIGHT command in Hot Screen, 3-8

- RIGHTGAP, 10-42
- RIGHTMARGIN parameter, 10-15
- row titles, 16-7
- row totals, 1-10, 7-2 to 7-3
  - calculated values and, 7-2, 7-6
  - calculating, 7-2 to 7-3
  - renaming, 7-2, 7-5
- rows, 1-7
  - assigning titles, 16-7
  - calculating percent, 2-19 to 2-20
  - calculating totals, 7-2 to 7-3
  - calculations and, 16-12 to 16-13
  - creating, 16-5 to 16-6, 16-8
  - identifying, 16-13 to 16-15
  - repeating, 16-15
  - saving, 16-33
  - sorting by, 1-9, 4-3 to 4-5
  - suppressing, 16-31 to 16-32
- ROW-TOTAL phrase, 7-2 to 7-3
- RPCT prefix operator, 2-19 to 2-20
- RUN command, 1-5

## S

- SALES data source, A-8 to A-9
- sample data sources, A-2
  - CAR, A-11
  - COMASTER, A-21
  - COURSES, A-16
  - creating, A-1, A-2
  - EDUCFILE, A-7
  - EMPDATA, A-17
  - EMPLOYEE, A-3
  - EXPERSON, A-18
  - FINANCE, A-14
  - Gotham Grinds data sources, A-29
  - JOBFILE, A-6
  - LEDGER, A-13
  - MOVIES, A-24
  - PAYHIST, A-20
  - PROD, A-10
  - REGION, A-15
  - SALES, A-8 to A-9

sample data sources (*continued*)

TRAINING, A-19

VIDEOTR2, A-26

VideoTrk, A-24

SAVB command, 11-23

SAVB files, 11-23

creating, 11-25 to 11-26

formatting, 11-25

SAVE command, 11-2, 11-23

SAVE files, 11-23

creating, 11-23 to 11-24

formatting, 11-23

GRAPH, 18-51

in Hot Screen, 3-9

SAVE formats, 11-26

ALPHA, 11-27

COM, 11-29

COMMA, 11-28

COMT, 11-30

DIF, 11-31

EXCEL, 11-31

HTML, 11-32

HTMTABLE, 11-33

LOTUS, 11-34

PDF, 11-34

WP, 11-38

SAVEMATRIX parameter, 15-7

saving intermediate report results, 16-32

saving report output, 1-24, 11-1 to 11-2

saving rows, 16-33

saving selected data, 3-9

scalar functions, 19-15

scatter diagrams, 18-35

SCREEN parameter, 3-3

screening conditions, 5-32

scrolling in Hot Screen, 3-7

SEG. operator, 1-16

segment instances, 12-1

missing descendants, 12-17 to 12-18

missing values and, 12-13 to 12-15

segment locations, 6-8

segment types, 2-24 to 2-25

segments, 1-16, 5-5, 12-3

missing instances, 12-3

MULTIPATH parameter and, 5-5, 5-9

screening, 13-31

SEGTYPE parameter, 11-21

selecting records, 1-10, 5-1, 5-5 to 5-6, 5-10 to 5-13, 5-15 to 5-16, 5-19 to 5-20, 5-26, 5-28 to 5-29

aggregate values and, 5-10

multi-segment data sources and, 5-25 to 5-26

VSAM data sources and, 5-39

selecting sort procedures, 4-34

selection criteria, 5-1 to 5-6, 5-13, 5-15 to 5-16, 7-21

partitioned FOCUS data sources and, 5-3

reading from a file, 5-29 to 5-32

selection of records

in free-form reports, 17-8

SEQUENCE, 10-40

-SET command, 8-2

SET END command in GRAPH, 18-60

SET parameters, 9-27 to 9-28, 11-14, 15-4, 19-21

ALL, 12-17 to 12-18

ASNAMES, 11-14 to 11-17

AUTOINDEX, 15-4

AUTOPATH, 15-4

BYPANEL, 3-12

BYSCROLL, 3-9

COLUMNS, 3-14

COMPUTE, 6-10

COUNTWIDTH, 2-10, 2-13

EMPTYREPORT, 3-6, 9-52

ESTRECORDS, 4-38 to 4-39

EXTAGGR, 4-35

EXTHOLD, 4-37 to 4-38

EXTSORT, 4-33 to 4-34



SET parameters (*continued*)

- FIELDNAME, 1-15
- FILE, 1-4
- FILTER, 5-32, 5-35
- FIXRETRIEVE, 11-21
- GRAPH, 18-60
- HOLDATTR, 11-14, 11-19 to 11-20
- HOLDLIST, 11-14, 11-18 to 11-19
- JOINOPT, 13-22
- ONLINE-FMT, 10-12
- PAGE, 9-5
- PANEL, 3-15
- PRINTPLUS, 3-4
- QUALTITLES, 9-17
- SAVEMATRIX, 15-7
- SCREEN, 3-3
- SPACES, 9-25
- SQLTOPTTF, 19-21
- STYLESHEET (STYLE), 10-11
- XRETRIEVAL, 3-12

short path definitions, 12-15

simple moving average, 6-21 to 6-22, 6-24

SIZE parameter, 10-19

SKIP-LINE command, 9-8 to 9-9

SKIPLINE phrase, 10-24

sort

- in free-form reports, 17-8

sort field values, 4-24

- hiding, 4-29
- ranking, 4-27 to 4-28
- restricting, 4-24 to 4-25, 4-27 to 4-28

sort fields, 4-1

- multi-path data sources and, 4-3, 4-6
- multiple, 4-4, 4-7, 4-31 to 4-32
- ranking values, 4-24
- temporary, 4-3, 4-5 to 4-6

sort order, 4-4, 4-7, 4-10

- grouping numeric data, 4-16 to 4-19
- specifying, 4-10 to 4-15

sort phrases, 2-3

sort procedures, 4-33

- querying, 4-34
- selecting, 4-34

sort sequence, 4-3, 4-6

sort values, 4-2 to 4-3, 4-16, 4-29

sorting, 4-33

- optimizing, 4-33

sorting by columns, 4-5 to 4-6

sorting by multiple fields, 4-4, 4-7

sorting by rows, 4-3 to 4-4

sorting report rows, 1-9

sorting reports, 6-15

SORTWORK files, 4-38 to 4-39

SPACES parameter, 9-25

special characters in free-form reports, 17-7

splits, 9-5

- preventing, 9-5 to 9-6

spot markers, 9-39

SQL format, 11-35

SQL join structures, 19-8 to 19-9

- qualified field names and, 19-10

SQL SELECT statement, 19-7

SQL statements, 19-2, 19-4

- FOCUS TABLE requests and, 19-1

SQL Translation Services, 19-2, 19-4

SQL Translator, 19-1

- aliases and, 19-10
- Cartesian product answer sets and, 19-13
- Continental Decimal Notation (CDN) and, 19-13
- CREATE TABLE command and, 19-11
- CREATE VIEW command and, 19-12
- date formats and, 19-15 to 19-16
- date-time values and, 19-17 to 19-19
- DELETE command and, 19-22
- DROP VIEW command and, 19-12 to 19-13
- field names and, 19-14
- generating TABLEF commands, 19-21

**SQL Translator (*continued*)**

- index optimized retrieval and, 19-20
- INSERT command and, 19-22
- INSERT INTO command and, 19-11
- JOIN command and, 19-8 to 19-10
- join structures and, 19-20
- reserved words and, 19-4
- SQLTOPTTF parameter and, 19-21
- time and timestamp fields and, 19-15 to 19-16
- UPDATE command and, 19-22

**SQL Translator commands, 19-5, 19-7**

- formatting commands and, 19-6

SQLDBC format, 11-35

SQLINF format, 11-36

SQLMSS format, 11-36

SQLODBC format, 11-36

SQLORA format, 11-37

SQLSYB format, 11-37

SQLTOPTTF parameter, 19-21

SQUEEZE parameter, 10-15

STAT query, 4-34

stoplighting, 10-49

structure diagrams, A-1 to A-2

STYLE, 10-11, 10-19

style sheets, 16-30

- financial reports and, 16-30

styled reports, 10-8

STYLESHEET, 10-11

StyleSheets, 10-1, 10-8

- ACROSS, 10-34
- ACROSSCOLUMN, 10-34
- ACROSSTITLE, 10-25
- ACROSSVALUE, 10-25
- activating, 10-11
- BOTTOMMARGIN, 10-14
- BY, 10-26
- CHECK STYLE, 10-19
- COLOR, 10-19

**StyleSheets (*continued*)**

- COLUMN, 10-34
- conditional styling, 10-49
- creating within a report request, 10-9
- DATA, 10-24
- FOCSTYLE file, 10-17
- FONT, 10-19
- FOOTING, 10-28
- GRANDTOTAL, 10-26
- HEADING, 10-28
- hierarchy, 10-43
- identifying report components, 10-21
- inheritance, 10-43
- ITEM, 10-29
- LEFTGAP, 10-42
- LEFTMARGIN, 10-15
- LINE, 10-29
- OBJECT, 10-29
- ORIENTATION, 10-14
- page layout parameters, 10-13
- PAGENUM, 10-24
- PAGESIZE, 10-16
- POSITION, 10-38
- printing, 10-12
- RECAP, 10-26
- REPORT, 10-24
- report components, 10-21
- requirements, 10-6 to 10-7
- RIGHTGAP, 10-42
- RIGHTMARGIN, 10-15
- SEQUENCE, 10-40
- SIZE, 10-19
- SKIPLINE, 10-24
- SQUEEZE, 10-15
- stoplighting, 10-49
- STYLE, 10-19
- StyleSheet file, 10-17
- SUBFOOT, 10-28
- SUBHEAD, 10-28
- SUBTOTAL, 10-26
- TABFOOTING, 10-28
- TABHEADING, 10-28
- TITLE, 10-25
- TOPMARGIN, 10-14
- UNDERLINE, 10-24
- UNITS, 10-14
- WHEN, 10-49, 10-51

- StyleSheets file, 10-17
- SUBFOOT command, 9-36 to 9-38
- subfootings, 9-36, 9-38, 10-28
  - creating, 9-37
  - displaying, 9-48, 9-50
  - inserting data in, 9-41, 9-43 to 9-44
- subheadings, 9-35 to 9-36, 10-28
  - creating, 9-35
  - displaying, 9-49
  - inserting data in, 9-41, 9-44
- subroutines, 16-22
  - calling, 16-22 to 16-23
- SUBTOTAL command, 7-8 to 7-11
- SUB-TOTAL command, 7-8 to 7-11
- subtotals, 1-10, 1-11, 7-1, 7-6, 7-16
  - calculated values and, 7-12, 7-14 to 7-15
  - COMPUTE command and, 7-16
  - displaying, 7-6 to 7-12, 9-48
  - RECAP command and, 7-16 to 7-19
  - SUBTOTAL, 10-26
- subtraction operator, 8-4 to 8-5
- SUM command, 2-1, 2-9 to 2-10, 4-2
  - merging data sources and, 14-12
- SUM prefix operator, 2-27
- SUMMARIZE command, 7-12 to 7-14
- summary lines, 7-21
  - displaying, 7-21
  - SUBFOOT, 10-28
- summing values, 4-2
- SUMPREFIX parameter, 4-37
- supplying data directly in FML, 16-24
- suppressing field display, 9-11 to 9-13
- suppressing field padding, 11-42 to 11-43
- suppressing grand totals, 7-19 to 7-20
- suppressing rows, 16-31 to 16-32
- suppressing sort field values, 4-29

- SUP-PRINT command, 4-29, 9-11, 9-13

- SYLK format, 11-37

- symbols, E-3
  - printable characters, E-3

- SyncSort sorting product, 4-33

- syntax summary, C-1
  - GRAPH, 18-57
  - TABLE, C-2

- system variables, 9-4
  - TABPAGENO, 9-4

## T

- tab-delimited extract files, 11-38

- TABFOOTING, 10-28

- TABHEADING, 10-28

- TABLE, C-2
  - basic reporting concepts, 3-6
  - displaying reports in Hot Screen, 3-1
  - displaying reports in TOE, 3-17
  - displaying reports with parameter set to ONLINE, 3-1
  - empty reports, 3-6
  - extracting data, 3-17
  - extracting data from Hot Screen, 3-9
  - help information, 3-6
  - OFFLINE, 3-1, 3-16
  - ONLINE, 3-1
  - previewing reports, 3-11 to 3-12
  - printing reports, 3-1
  - redefining field formats, 3-11
  - redisplaying reports, 3-11, 3-16
  - StyleSheets, 10-1
  - suppressing report display, 3-6
  - with zero records, 3-6

- TABLE FILE command, 1-3

- TABLE requests, 19-1
  - SQL statements and, 19-1

- TABLEF command, 15-6, 19-21
  - data retrieval and, 15-6
  - SQL Translator and, 19-21

TABPAGENO variable, 9-4

TABT format, 11-38

tag names, 13-9

tagged rows, 16-31 to 16-32  
  suppressing, 16-31 to 16-32

Tektronics terminals, 18-57

temporary fields, 1-12, 6-1, 6-3  
  as sort fields, 4-3, 4-6  
  creating, 1-12  
  creating with COMPUTE, 6-2, 6-11 to 6-13  
  creating with DEFINE, 6-2 to 6-4  
  creating with DEFINE FUNCTION, 6-33  
  missing values and, 12-6  
  subroutines and, 6-32

Terminal Operator Environment (TOE), 3-17

TERMINAL parameter, 18-60

testing for blanks or zeros, 12-11

testing for existing data, 12-10

testing for missing segment instances, 12-19

testing for missing values, 12-9

text, 9-39  
  in free-form reports, 17-6  
  positioning, 9-40

text fields, 4-3  
  output files and, 11-26, 11-40 to 11-41

tick intervals in GRAPH, 18-42

TILE column, 4-19 to 4-20, 4-22, 4-23

tile fields, 4-19 to 4-20, 4-22 to 4-23

TILES phrase, 4-19 to 4-20, 4-22 to 4-23

time fields, 19-15  
  SQL Translator and, 19-16

timestamp fields, 19-15  
  SQL Translator and, 19-16

TITLE attribute, 11-19 to 11-20

TITLE parameter, 10-25

TO phrase, 16-9

Creating Reports

TOP command in Hot Screen, 3-7

TOPMARGIN parameter, 10-14

TOT prefix operator, 2-27 to 2-28

totaling rows and columns, 1-10

totals, 1-10 to 1-11, 4-7 to 4-8, 7-1  
  suppressing, 7-19

TRAINING data source, A-19

trend values, 6-17

TYPE attribute in StyleSheet, 10-21  
  ACROSSTITLE, 10-25  
  ACROSSVALUE, 10-25  
  DATA, 10-24  
  FOOTING, 10-28  
  GRANDTOTAL, 10-26  
  HEADING, 10-28  
  PAGENUM, 10-24  
  RECAP, 10-26  
  REPORT, 10-24  
  SKIPLINE, 10-24  
  SUBFOOT, 10-28  
  SUBHEAD, 10-28  
  SUBTOTAL, 10-26  
  TABFOOTING, 10-28  
  TABHEADING, 10-28  
  TITLE, 10-25  
  UNDERLINE, 10-24

## U

UNDER-LINE command, 9-10

UNDERLINE phrase, 10-24

UNION operator, 19-15

unique join structures, 13-2, 13-4 to 13-5, 13-15

UNITS parameter, 10-14

universal concatenation, 14-14, 18-17  
  field names and, 14-17 to 14-18  
  MORE phrase, 18-17  
  MORE phrase and, 14-14 to 14-16

UP command in Hot Screen, 3-7

UPDATE command, 19-22

user-coded programs, D-1

## V

values, 9-41

    embedding, 9-41

VAUTO parameter, 18-43, 18-60

VAXIS parameter, 18-43, 18-60

VCLASS parameter, 18-44, 18-60

verbs, 2-1, 4-2

    multiple, 4-31

vertical axis features, 18-60

    class and tick intervals, 18-42, 18-44, 18-60

    graph element, 18-7

    grids, 18-7, 18-45, 18-60

    height, 18-43

    scale, 18-43

VGRID parameter, 18-37, 18-45, 18-60

VIDEOTR2 data source, A-26

VideoTrk data source, A-24

virtual fields, 6-2 to 6-3

    adding, 6-7, 6-8

    associating segments, 6-8

    creating, 6-4, 6-6 to 6-7, 6-9

    deleting, 6-4, 6-6 to 6-7, 6-8

    filters and, 5-32, 5-34

    in Master Files, 6-4, 6-6

    join structures and, 6-4, 6-6, 6-11, 13-17, 13-19

        to 13-20, 13-26 to 13-27, 13-29

    MISSING attribute and, 12-5

    missing values and, 12-6

    saving, 6-10 to 6-11, 13-26 to 13-27, 13-29

VMAX parameter, 18-43, 18-60

VMIN parameter, 18-43, 18-60

VMSORT sorting product, 4-33

VSAM data sources, 5-39

    selecting records and, 5-39

VTICK parameter, 18-44, 18-60

VZERO parameter, 18-48, 18-60

## W

WebFOCUS StyleSheets, 16-30

    formatting financial reports and, 16-30

WHEN clause, 9-46 to 9-47, 9-50

WHEN command, 10-49

    ACROSSCOLUMN, 10-51

WHEN EXISTS phrase, 16-32

WHEN phrase, 7-21, 8-2

    expressions and, 8-2

WHERE operator, 5-13, 5-15, 5-17 to 5-19

WHERE phrase, 5-2, to 5-4, 5-10 to 5-12, 5-20, 5-22, 5-29, 8-2

    existing data and, 12-10

    expressions and, 8-2

    missing values and, 12-9, 12-11

    selecting records and, 5-30 to 5-31

WHERE TOTAL phrase, 5-10 to 5-11

    COMPUTE and, 6-16

WHERE-based joins, 13-2

widow lines, 9-5 to 9-6

    preventing, 9-6

wildcard characters, 5-20

    relational operators and, 5-20, 5-22

    treating as literals, 5-24 to 5-25

WITHIN phrase, 2-29

WP format, 11-38

WRITE command, 2-9

## Z

zeros, 12-11

    testing for, 12-11

---

# Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. You can contact us through the following methods:

**Mail:** Documentation Services – Customer Support  
Information Builders, Inc.  
Two Penn Plaza  
New York, NY 10121-2898

**Fax:** (212) 967-0460

**E-mail:** [books\\_info@ibi.com](mailto:books_info@ibi.com)

**Web form:** <http://www.informationbuilders.com/bookstore/derf.html>

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

Telephone: \_\_\_\_\_ Date: \_\_\_\_\_

E-mail: \_\_\_\_\_

Comments:

---

## Reader Comments