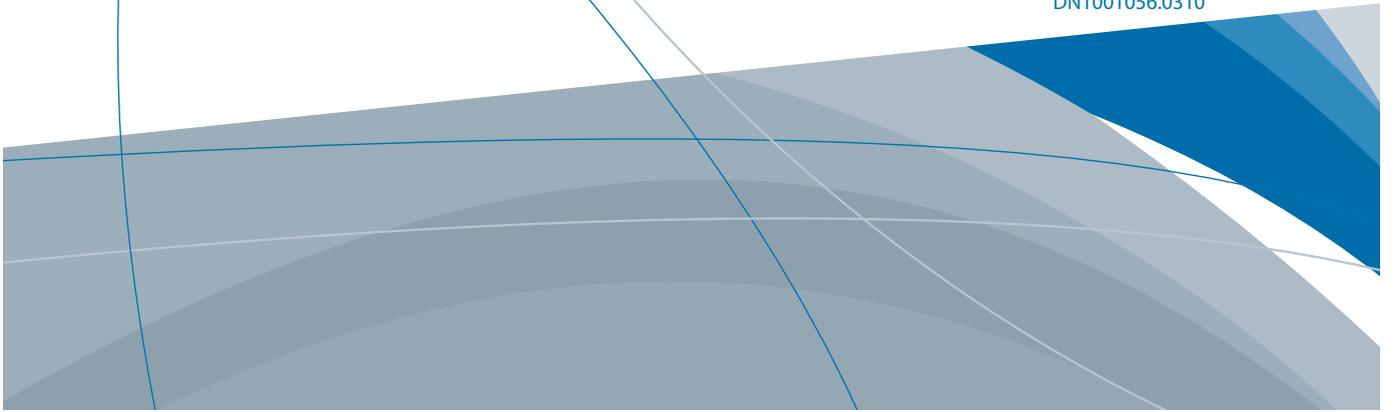


FOCUS for Mainframe **Creating Reports**

Version 7.6

DN1001056.0310

A decorative graphic at the bottom of the page consisting of overlapping, semi-transparent geometric shapes in various shades of blue and grey, creating a layered, abstract effect.

Cactus, EDA, EDA/SQL, FIDEL, FOCUS, Information Builders, the Information Builders logo, iWay, iWay Software, Parlay, PC/FOCUS, RStat, TableTalk, Web390, and WebFOCUS are registered trademarks, and Magnify is a trademark of Information Builders, Inc.

Due to the nature of this material, this document refers to numerous hardware and software products by their trademarks. 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 © 2010, by Information Builders, Inc. and iWay Software. All rights reserved. Patent Pending. This manual, or parts thereof, may not be reproduced in any form without the written permission of Information Builders, Inc.

Contents

Preface.....	23
Documentation Conventions.....	26
Related Publications.....	27
Customer Support.....	27
Information You Should Have.....	27
User Feedback.....	28
Information Builders Consulting and Training.....	29
1. Creating Tabular Reports.....	31
Requirements for Creating a Report.....	32
Creating a Report Request.....	33
Beginning a Report Request.....	33
Requesting Help When Issuing a Report Request.....	35
Completing a Report Request.....	35
Selecting a Report Output Destination.....	36
Developing Your Report Request.....	36
Including Display Fields in a Report Request.....	39
Referring to Fields in a Report Request.....	40
Referring to an Individual Field.....	40
Referring to Fields Using Qualified Field Names.....	41
Referring to All of the Fields in a Segment.....	43
Displaying a List of Field Names.....	43
Listing Field Names, Aliases, and Format Information.....	44
2. Displaying Report Data.....	45
Using Display Commands in a Request.....	46
Displaying Individual Values.....	47
Displaying All Fields.....	49
Displaying All Fields in a Segment.....	49
Displaying the Structure and Retrieval Order of a Multi-Path Data Source.....	51

Adding Values.....	53
Counting Values.....	55
Counting Segment Instances.....	56
Expanding Byte Precision for COUNT and LIST.....	57
Maximum Number of Display Fields Supported in a Request.....	59
Manipulating Display Fields With Prefix Operators	60
Prefix Operator Basics.....	61
Averaging Values of a Field.....	63
Averaging the Sum of Squared Fields.....	63
Calculating Maximum and Minimum Field Values.....	64
Calculating Column and Row Percents.....	64
Producing a Direct Percent of a Count.....	66
Aggregating and Listing Unique Values.....	67
Retrieving First and Last Records.....	69
Summing and Counting Values.....	71
Ranking Sort Field Values With RNK.....	73
Changing the Format of a Report Column.....	76
Determining the Width of a Report Column.....	78

3. Viewing and Printing Report Output.....81

Displaying Reports in Hot Screen.....	82
Using PRINTPLUS.....	83
Accessing Help Information.....	84
Scrolling a Report.....	85
Scrolling Forward.....	85
Scrolling Backward.....	86
Scrolling Horizontally.....	86
Scrolling From Fixed Columns (Fencing).....	87
Scrolling Report Headings.....	87
Saving Selected Data.....	88
Locating Character Strings.....	88
Repeating Commands.....	88
Redisplaying Reports.....	89
Previewing Your Report.....	90

Displaying BY Fields With Panels.....	90
Scrolling by Columns of BY Fields.....	92
The SET COLUMNS Command.....	93
Displaying Reports in the Panel Facility.....	93
Printing Reports.....	94
The OFFLINE Command.....	94
Printing Reports in Hot Screen.....	95
Displaying Reports in the Terminal Operator Environment.....	95
4. Sorting Tabular Reports.....	97
Sorting Tabular Reports Overview.....	98
Sorting Rows.....	99
Displaying All Vertical (BY) Sort Field Values.....	101
Using Multiple Vertical (BY) Sort Fields.....	102
Displaying a Row for Data Excluded by a Sort Phrase.....	103
Sorting Columns.....	106
Controlling Underlines for ACROSS Objects.....	108
Using Multiple Horizontal (ACROSS) Sort Fields.....	110
Collapsing PRINT With ACROSS.....	111
Manipulating Display Field Values in a Sort Group.....	113
Creating a Matrix Report.....	115
Specifying the Sort Order.....	116
Specifying Your Own Sort Order.....	119
Ranking Sort Field Values.....	122
Grouping Numeric Data Into Ranges.....	124
Grouping Numeric Data Into Tiles.....	127
Restricting Sort Field Values by Highest/Lowest Rank.....	133
Sorting and Aggregating Report Columns.....	135
Restricting the Number of Columns in a Report.....	137
Hiding Sort Values.....	138
Sorting With Multiple Display Commands.....	139
Controlling Formatting of Reports With Multiple Display Commands.....	141
Improving Efficiency With External Sorts.....	146
Providing an Estimate of Input Records or Report Size for Sorting.....	148

Mainframe External Sort Utilities and Message Options.....	149
Aggregation by External Sort (Mainframe Environments Only).....	153
Changing Retrieval Order With Aggregation	155
Creating a HOLD File With an External Sort	155
5. Selecting Records for Your Report.....	157
Selecting Records Overview.....	158
Choosing a Filtering Method.....	158
Selections Based on Individual Values.....	159
Controlling Record Selection in Multi-path Data Sources.....	162
Selection Based on Aggregate Values.....	167
Using Compound Expressions for Record Selection.....	169
Using Operators in Record Selection Tests.....	171
Types of Record Selection Tests.....	174
Range Tests With FROM and TO.....	175
Range Tests With GE and LE or GT and LT.....	176
Missing Data Tests.....	178
Character String Screening With CONTAINS and OMITS.....	179
Screening on Masked Fields With LIKE and IS.....	180
Using an Escape Character for LIKE.....	184
Qualifying Parent Segments Using INCLUDES and EXCLUDES.....	187
Selections Based on Group Key Values.....	188
Setting Limits on the Number of Records Read.....	189
Selecting Records Using IF Phrases.....	190
Reading Selection Values From a File.....	191
Assigning Screening Conditions to a File.....	195
Preserving Filters Across Joins.....	201
VSAM Record Selection Efficiencies.....	204
Reporting From Files With Alternate Indexes.....	204
6. Creating Temporary Fields.....	205
What Is a Temporary Field?.....	206
Defining a Virtual Field.....	209
Defining Multiple Virtual Fields.....	215
Displaying Virtual Fields.....	216

Clearing a Virtual Field.....	216
Establishing a Segment Location for a Virtual Field.....	217
Defining Virtual Fields Using a Multi-Path Data Source.....	218
Increasing the Speed of Calculations in Virtual Fields.....	219
Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN.....	219
Applying Dynamically Formatted Virtual Fields to Report Columns.....	220
Creating a Calculated Value.....	224
Using Positional Column Referencing With Calculated Values.....	227
Using ACROSS With Calculated Values.....	228
Sorting Calculated Values.....	229
Screening on Calculated Values.....	229
Assigning Column Reference Numbers.....	229
Using Column Notation in a Report Request.....	230
Calculating Trends and Predicting Values With FORECAST.....	239
FORECAST Processing.....	241
Using a Simple Moving Average.....	245
Using Single Exponential Smoothing.....	248
Using Double Exponential Smoothing.....	251
Using Triple Exponential Smoothing.....	252
Using a Linear Regression Equation.....	254
FORECAST Reporting Techniques.....	257
Calculating Trends and Predicting Values With Multivariate REGRESS.....	260
Using Text Fields in DEFINE and COMPUTE.....	263
Creating Temporary Fields Independent of a Master File.....	264
7. Including Totals and Subtotals.....	269
Calculating Row and Column Totals.....	270
Producing Row Totals for Horizontal (ACROSS) Sort Field Values.....	276
Including Section Totals and a Grand Total.....	278
Including Subtotals.....	279
Recalculating Values for Subtotal Rows.....	283
Manipulating Summary Values With Prefix Operators.....	287
Controlling Summary Line Processing.....	295
Using Prefix Operators With Calculated Values.....	302

Using Multiple SUB-TOTAL or SUMMARIZE Commands With Prefix Operators.....	304
Combinations of Summary Commands.....	306
Producing Summary Columns for Horizontal Sort Fields.....	313
Performing Calculations at Sort Field Breaks.....	315
Suppressing Grand Totals.....	319
Conditionally Displaying Summary Lines and Text.....	320
8. Using Expressions.....	323
Using Expressions in Commands and Phrases.....	324
Types of Expressions.....	325
Expressions and Field Formats.....	326
Creating a Numeric Expression.....	327
Order of Evaluation.....	330
Creating a Date Expression.....	331
Formats for Date Values.....	332
Performing Calculations on Dates.....	334
Cross-Century Dates With DEFINE and COMPUTE.....	335
Returned Field Format Selection.....	335
Using a Date Constant in an Expression.....	335
Extracting a Date Component.....	336
Combining Fields With Different Formats in an Expression.....	336
Creating a Date-Time Expression.....	337
Specifying a Date-Time Value.....	337
Manipulating Date-Time Values.....	342
Creating a Character Expression.....	344
Embedding a Quotation Mark in a Quote-Delimited Literal String.....	344
Concatenating Character Strings.....	345
Creating a Variable Length Character Expression.....	347
Using Concatenation With AnV Fields.....	347
Using the EDIT Function With AnV Fields.....	348
Using CONTAINS and OMITS With AnV Fields.....	348
Using LIKE With AnV Fields.....	348
Using the EQ, NE, LT, GT, LE, and GE Operators With AnV Fields.....	349
Using the DECODE Function With AnV Fields.....	350

Using the Assignment Operator With AnV Fields.....	350
Creating a Logical Expression.....	351
Creating a Conditional Expression.....	353
9. Customizing Tabular Reports.....	357
Producing Headings and Footings.....	358
Limits for Headings and Footings.....	359
Report and Page Headings.....	359
Report and Page Footings.....	363
Subheads and Subfoots.....	365
Using Data in Headings and Footings.....	372
Positioning Text.....	374
Extending Heading and Footing Code to Multiple Lines.....	376
Producing a Free-Form Report.....	380
Creating Paging and Numbering.....	380
Specifying a Page Break: PAGE-BREAK.....	381
Inserting Page Numbers: TABPAGENO.....	382
Controlling Report Page Numbering.....	383
Suppressing Page Numbers: SET PAGE.....	387
Preventing an Undesirable Split: NOSPLIT.....	387
Suppressing Fields: SUP-PRINT or NOPRINT.....	391
Reducing a Report's Width: FOLD-LINE and OVER.....	394
Compressing the Columns of Reports: FOLD-LINE.....	395
Decreasing the Width of a Report: OVER.....	396
Positioning Columns: IN.....	398
Separating Sections of a Report: SKIP-LINE and UNDER-LINE.....	401
Adding Blank Lines: SKIP-LINE.....	401
Underlining Values: UNDER-LINE.....	403
Controlling Column Spacing: SET SPACES.....	405
Creating New Column Titles: AS.....	406
Customizing Column Names: SET QUALTITLES.....	408
Column Title Justification.....	409
Customizing Reports With SET Parameters.....	410
Conditionally Formatting Reports With the WHEN Clause.....	411

Controlling the Display of Empty Reports.....	419
10. Saving and Reusing Your Report Output.....	421
Saving Your Report Output.....	422
Naming and Storing Report Output Files.....	422
Creating a HOLD File.....	423
Holding Report Output in FOCUS Format.....	429
Controlling Attributes in HOLD Master Files.....	434
Controlling Field Names in a HOLD Master File.....	435
Controlling Fields in a HOLD Master File.....	439
Controlling the TITLE and ACCEPT Attributes in the HOLD Master File.....	441
Keyed Retrieval From HOLD Files.....	443
Using DBMS Temporary Tables as HOLD Files.....	445
Column Names in the HOLD File.....	448
Primary Keys and Indexes in the HOLD File.....	449
Creating SAVE and SAVB Files.....	449
Creating a PCHOLD File.....	453
Choosing Output File Formats.....	455
Using Text Fields in Output Files.....	472
Creating a Delimited Sequential File.....	473
Saving Report Output in INTERNAL Format.....	478
Creating a Structured HOLD File.....	481
11. Styling Reports.....	491
Introduction to Styled Reports.....	492
Choosing a Type of Style Sheet.....	495
Choosing an Output Format.....	496
Styling Reports With StyleSheets.....	499
What Is a StyleSheet?.....	500
What Is a Style?.....	501
Comparison of Reports With and Without StyleSheets.....	501
Creating a StyleSheet.....	503
StyleSheet Syntax.....	506
Improving FOCUS StyleSheet Readability.....	507
Adding a Comment to a FOCUS StyleSheet.....	507

Checking StyleSheet Syntax.....	508
Creating a Styled Report.....	508
Styling the Page Layout.....	509
Selecting Page Size, Orientation, and Color.....	510
Setting Page Margins.....	515
Displaying Current Settings: The ? SET STYLE Query.....	517
Specifying Font Format in a Report.....	518
Specifying Fonts for Reports.....	523
Identifying Report Components.....	525
Identifying an Entire Report, Column, or Row.....	527
Identifying Data.....	537
Identifying Totals and Subtotals.....	542
Identifying a Heading, Footing, Title, or FML Free Text.....	548
Identifying a Column or Row Title.....	549
Identifying a Heading or Footing.....	554
Identifying a Page Number, Underline, or Skipped Line.....	565
Reusing FOCUS StyleSheet Declarations With Macros.....	578
Defining a FOCUS StyleSheet Macro.....	579
Applying a FOCUS StyleSheet Macro.....	579
FOCUS StyleSheet Attribute Inheritance.....	581
Conditionally Formatting in a StyleSheet.....	585
Applying Sequential Conditional Formatting.....	586
Using Conditional Grid Formatting in a Field.....	603
12. Cascading Style Sheets.....	605
What Are Cascading Style Sheets?.....	606
Benefits of Cascading Style Sheets.....	606
The Notion of Browser Dependence.....	607
Types of Cascading Style Sheets.....	607
Cascading Style Sheets and Precedence Rules.....	608
Cascading Style Sheet Formatting Statements: Rules and Classes.....	609
Selecting a CSS Rule.....	609
Naming CSS Classes.....	610
Inheritance and CSS.....	610

Generating an Internal Cascading Style Sheet.....	612
Selecting a Unit of Measurement.....	613
Working With External Cascading Style Sheets.....	614
Applying CSS Styles.....	614
Using an External CSS - A Graphical Overview.....	616
Combining CSS Styling With Other Formatting Methods.....	620
Combining an External CSS With a FOCUS StyleSheet.....	620
Combining an External CSS With TABLE Language Formatting.....	622
Linking to an External Cascading Style Sheet.....	622
FAQ About Using External Cascading Style Sheets.....	625
Troubleshooting Cascading Style Sheets.....	626

13. Working With Styled Output Formats.....629

Working With HTML Reports.....	630
Preserving Leading and Internal Blanks in Report Output.....	630
Creating HTML Reports With Absolute Positioning.....	632
Working With Excel 2000 and Excel 97 Reports.....	635
Creating Styled Excel 2000 Files.....	636
National Language Support With EXL2K.....	638
Displaying Formatted Dates and Numeric Values.....	639
Controlling Column Width and Wrapping.....	644
Locking Columns in Excel Report Output.....	647
Using the Excel 2000 Formula Option	651
Using the Excel 2000 PIVOT Option	657
Designating CACHEFIELDS in PivotTables.....	661
Designating PAGEFIELDS in PivotTables.....	663
Excel Named Ranges.....	665
Identifying Null Values in Excel 2000.....	667
Excel Table of Contents.....	670
Excel Compound Reports.....	672
Transferring Excel 2000 Formatted Files Using FTP.....	684
Creating Styled Excel 97 Files.....	685
Working With PostScript and PDF Reports.....	687
Creating Compound PDF or PostScript Reports.....	689

Adding PostScript Type 1 Fonts for PS and PDF Formats.....	692
Creating PDF Files for Use With UNIX Systems.....	699
Displaying An and AnV Fields With Line Breaks.....	701
14. Advanced StyleSheet Features.....	705
Positioning a Report Component.....	706
Arranging Pages and Columns on a Page.....	717
Determining Column Width.....	719
Wrapping and Justifying Report Components.....	722
Controlling Wrapping of Report Data.....	723
Controlling Wrapping With Alternative Methods.....	728
Justifying Report Columns.....	732
Justifying a Heading or Footing.....	734
Justifying a Column Title.....	737
Justifying a Label for a Subtotal or Grand Total.....	740
Aligning Heading and Footing Elements.....	742
Aligning a Heading or Footing Element in an HTML Report.....	744
Aligning a Heading or Footing Element Across Columns in an HTML Report.....	747
Aligning Content in a Multi-Line Heading or Footing.....	755
Aligning Decimals in a Multi-Line Heading or Footing.....	760
Combining Column and Line Formatting in Headings and Footings.....	762
Adding Grids and Borders.....	767
Adding an Image to a Report.....	773
Linking in a Report.....	785
Linking to a URL.....	786
Linking to a JavaScript Function.....	788
Linking With Conditions.....	789
Linking From a Graphic Image.....	791
Specifying a Base URL.....	793
Specifying a Target Frame.....	794
Linking Report Pages.....	796
Working With Mailing Labels and Multi-Pane Pages.....	801
15. Handling Records With Missing Field Values.....	807
Irrelevant Report Data.....	808

Missing Field Values.....	809
MISSING Attribute in the Master File.....	810
MISSING Attribute in a DEFINE or COMPUTE Command.....	812
Testing for a Segment With a Missing Field Value.....	815
Preserving Missing Data Values in an Output File.....	818
Propagating Missing Values to Reformatted Fields in a Request.....	820
Handling a Missing Segment Instance.....	822
Including Missing Instances in Reports With the ALL. Prefix.....	825
Including Missing Instances in Reports With the SET ALL Parameter.....	826
Testing for Missing Instances in FOCUS Data Sources.....	827
Setting the NODATA Character String.....	828
16. Joining Data Sources.....	831
Types of Joins.....	832
Unique and Non-Unique Joined Structures.....	835
Recursive Joined Structures.....	839
How the JOIN Command Works.....	843
Creating an Equijoin.....	845
Joining From a Virtual Field to a Real Field Using an Equijoin.....	855
Data Formats of Shared Fields.....	859
Joining Fields With Different Numeric Data Types.....	860
Using a Conditional Join.....	861
Preserving Virtual Fields During Join Parsing.....	865
Preserving Virtual Fields Using KEEPDEFINES.....	865
Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN.....	868
Screening Segments With Conditional JOIN Expressions.....	870
Parsing WHERE Criteria in a Join.....	870
Displaying Joined Structures.....	871
Clearing Joined Structures.....	874
Clearing a Conditional Join.....	874
17. Merging Data Sources.....	877
Merging Data	878
MATCH Processing.....	881
MATCH Processing With Common High-Order Sort Fields.....	884

Fine-Tuning MATCH Processing.....	888
Universal Concatenation.....	890
Field Name and Format Matching.....	893
Merging Concatenated Data Sources.....	895
Using Sort Fields in MATCH Requests.....	897
Cartesian Product.....	900
18. Improving Report Processing.....	903
Rotating a Data Structure for Enhanced Retrieval.....	904
Optimizing Retrieval Speed for FOCUS Data Sources.....	907
Automatic Indexed Retrieval.....	907
Data Retrieval Using TABLEF.....	910
Preserving the Internal Matrix of Your Last Report.....	911
Compiling Expressions.....	912
Compiling Expressions Using the DEFINES Parameter.....	912
Compiling Expressions Using the COMPUTE Parameter.....	914
Producing Multiple Outputs in One Pass of a Data Source (Pooled Tables).....	916
Overview.....	917
Sub-Pool Boundaries and Pooling Restrictions.....	919
Estimating Memory Requirements.....	922
Memory Requirements.....	924
Sharing Selection Criteria and Filters Across Requests in a Pool.....	928
Criteria When Pooling Non-Relational Database Requests.....	928
Criteria When Pooling Relational Database Requests.....	928
Criteria When Pooling Batch Requests.....	930
Selecting a Sort Utility.....	931
Observing the Results of Pooling (TRACEON).....	932
Installing the Pooled Tables Option.....	935
19. Creating Financial Reports With Financial Modeling Language (FML).....	937
Reporting With FML.....	938
Creating Rows From Data.....	941
Creating Rows From Multiple Records.....	944
Using the BY Phrase in FML Requests.....	950
Combining BY and FOR Phrases in an FML Request.....	951

Supplying Data Directly in a Request.....	952
Performing Inter-Row Calculations.....	953
Referring to Rows in Calculations.....	955
Referring to Columns in Calculations.....	958
Referring to Column Numbers in Calculations.....	959
Referring to Contiguous Columns in Calculations.....	960
Referring to Column Addresses in Calculations.....	961
Referring to Relative Column Addresses in Calculations.....	962
Applying Relative Column Addressing in a RECAP Expression.....	962
Controlling the Creation of Column Reference Numbers.....	962
Referring to Column Values in Calculations.....	964
Referring to Cells in Calculations.....	965
Using Functions in RECAP Calculations.....	966
Inserting Rows of Free Text.....	968
Adding a Column to an FML Report.....	971
Creating a Recursive Model.....	973
Reporting Dynamically From a Hierarchy.....	975
Requirements for FML Hierarchies.....	976
Displaying an FML Hierarchy.....	978
Consolidating an FML Hierarchy.....	981
Loading a Hierarchy Manually.....	988
Customizing a Row Title.....	990
Formatting an FML Report.....	992
Indenting Row Titles in an FML Hierarchy.....	997
Suppressing the Display of Rows.....	1001
Suppressing Rows With No Data.....	1002
Saving and Retrieving Intermediate Report Results.....	1002
Posting Data.....	1003
Creating HOLD Files From FML Reports.....	1006
20. Creating a Free-Form Report.....	1009
Creating a Free-Form Report.....	1010
Designing a Free-Form Report.....	1014
Incorporating Text in a Free-Form Report.....	1014

Incorporating Data Fields in a Free-Form Report.....	1015
Incorporating Graphic Characters in a Free-Form Report.....	1015
Laying Out a Free-Form Report.....	1016
Sorting and Selecting Records in a Free-Form Report.....	1016
21. Creating Graphs: GRAPH.....	1017
Introduction.....	1018
GRAPH vs. TABLE Requests.....	1018
Running Graph Requests Offline.....	1025
Controlling the Format.....	1026
Graphic Devices Supported.....	1029
Command Syntax.....	1031
GRAPH vs. TABLE Syntax.....	1031
Specifying Graph Forms and Contents.....	1033
Graph Forms.....	1040
Connected Point Plots.....	1041
Histograms.....	1045
Bar Charts.....	1048
Pie Charts.....	1053
Scatter Diagrams.....	1056
Adjusting Graph Elements.....	1058
The Horizontal Axis: System Defaults.....	1060
The Vertical Axis: System Defaults.....	1063
Highlighting Facilities.....	1065
Special Topics.....	1066
Plotting Dates.....	1067
Handling Missing Data.....	1069
Using Fixed-Axis Scales.....	1071
Saving Formatted GRAPH Output.....	1072
Creating Formatted Input for CA-TELLAGRAF.....	1074
Using the FOCUS ICU Interface.....	1074
Special Graphics Devices.....	1075
Medium-Resolution Devices.....	1075
High-Resolution Devices.....	1076

Command and SET Parameter Summary.....	1078
GRAPH Command.....	1079
SET Parameters.....	1081
22. Using SQL to Create Reports.....	1089
Supported and Unsupported SQL Statements.....	1090
Using SQL Translator Commands.....	1093
The SQL SELECT Statement.....	1095
SQL Joins.....	1096
SQL CREATE TABLE and INSERT INTO Commands.....	1099
SQL CREATE VIEW and DROP VIEW Commands.....	1100
SQL PREPARE, EXECUTE, and COMMIT Commands.....	1101
Cartesian Product Style Answer Sets.....	1102
Continental Decimal Notation (CDN).....	1102
Specifying Field Names in SQL Requests.....	1103
SQL UNION, INTERSECT, and EXCEPT Operators.....	1103
Numeric Constants, Literals, Expressions, and Functions.....	1104
SQL Translator Support for Date, Time, and Timestamp Fields.....	1104
Extracting Date-Time Components Using the SQL Translator.....	1106
Index Optimized Retrieval.....	1109
Optimized Joins.....	1109
TABLEF Optimization.....	1110
SQL INSERT, UPDATE, and DELETE Commands.....	1111
A. Master Files and Diagrams.....	1113
Creating Sample Data Sources.....	1114
EMPLOYEE Data Source.....	1116
EMPLOYEE Master File.....	1118
EMPLOYEE Structure Diagram.....	1119
JOBFILE Data Source.....	1120
JOBFILE Master File.....	1120
JOBFILE Structure Diagram.....	1121
EDUCFILE Data Source.....	1121
EDUCFILE Master File.....	1122
EDUCFILE Structure Diagram.....	1122

SALES Data Source.....1123
 SALES Master File.....1123
 SALES Structure Diagram.....1124
 PROD Data Source.....1125
 PROD Master File.....1125
 PROD Structure Diagram.....1125
 CAR Data Source.....1126
 CAR Master File.....1127
 CAR Structure Diagram.....1128
 LEDGER Data Source.....1129
 LEDGER Master File.....1129
 LEDGER Structure Diagram.....1129
 FINANCE Data Source.....1130
 FINANCE Master File.....1130
 FINANCE Structure Diagram.....1130
 REGION Data Source.....1131
 REGION Master File.....1131
 REGION Structure Diagram.....1131
 COURSES Data Source.....1132
 COURSES Master File.....1132
 COURSES Structure Diagram.....1132
 EMPDATA Data Source.....1133
 EMPDATA Master File.....1133
 EMPDATA Structure Diagram.....1133
 EXPERSON Data Source.....1134
 EXPERSON Master File.....1134
 EXPERSON Structure Diagram.....1135
 TRAINING Data Source.....1135
 TRAINING Master File.....1135
 TRAINING Structure Diagram.....1136
 COURSE Data Source.....1136
 COURSE Master File.....1136
 COURSE Structure Diagram.....1137
 JOBHIST Data Source.....1137

JOBHIST Master File.....	1137
JOBHIST Structure Diagram.....	1138
JOBLIST Data Source.....	1138
JOBLIST Master File.....	1138
JOBLIST Structure Diagram.....	1139
LOCATOR Data Source.....	1139
LOCATOR Master File.....	1139
LOCATOR Structure Diagram.....	1140
PERSINFO Data Source.....	1140
PERSINFO Master File.....	1140
PERSINFO Structure Diagram.....	1141
SALHIST Data Source.....	1141
SALHIST Master File.....	1141
SALHIST Structure Diagram.....	1141
PAYHIST File.....	1142
PAYHIST Master File.....	1142
PAYHIST Structure Diagram.....	1142
COMASTER File.....	1143
COMASTER Master File.....	1144
COMASTER Structure Diagram.....	1145
VIDEOTRK, MOVIES, and ITEMS Data Sources.....	1146
VIDEOTRK Master File.....	1146
VIDEOTRK Structure Diagram.....	1147
MOVIES Master File.....	1148
MOVIES Structure Diagram.....	1148
ITEMS Master File.....	1149
ITEMS Structure Diagram.....	1149
VIDEOTR2 Data Source.....	1150
VIDEOTR2 Master File.....	1150
VIDEOTR2 Structure Diagram.....	1151
Gotham Grinds Data Sources.....	1152
GGDEMOG Master File.....	1153
GGDEMOG Structure Diagram.....	1153
GGORDER Master File.....	1154

GGORDER Structure Diagram.....	1154
GGPRODS Master File.....	1155
GGPRODS Structure Diagram.....	1155
GGSALES Master File.....	1156
GGSALES Structure Diagram.....	1156
GGSTORES Master File.....	1157
GGSTORES Structure Diagram.....	1157
Century Corp Data Sources.....	1158
CENTCOMP Master File.....	1160
CENTCOMP Structure Diagram.....	1160
CENTFIN Master File.....	1161
CENTFIN Structure Diagram.....	1161
CENTHR Master File.....	1162
CENTHR Structure Diagram.....	1164
CENTINV Master File.....	1165
CENTINV Structure Diagram.....	1165
CENTORD Master File.....	1166
CENTORD Structure Diagram.....	1167
CENTQA Master File.....	1168
CENTQA Structure Diagram.....	1169
CENTGL Master File.....	1169
CENTGL Structure Diagram.....	1170
CENTSYSF Master File.....	1170
CENTSYSF Structure Diagram.....	1170
CENTSTMT Master File.....	1171
CENTSTMT Structure Diagram.....	1172
B. Error Messages.....	1173
Accessing Error Files.....	1174
Displaying Messages.....	1174
C. Table Syntax Summary.....	1177
TABLE Syntax Summary.....	1178
TABLEF Syntax Summary.....	1180
MATCH Syntax Summary.....	1181

FOR Syntax Summary.....	1182
D. Writing User-Coded Programs to Create HOLD Files.....	1183
Arguments Used in Calls to Programs That Create HOLD Files.....	1184
Reader Comments.....	1231

Preface

This documentation describes how to use FOCUS Version 7.6 in the z/VM and z/OS environments. It is intended for all FOCUS users. This manual is part of the FOCUS documentation set.

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

How This Manual Is Organized

This manual includes the following chapters:

	Chapter/Appendix	Contents
1	Creating Tabular Reports	Provides an introduction to the TABLE command, a powerful tool for analyzing data.
2	Displaying Report Data	Describes ways to retrieve field values from a data source and display them.

	Chapter/Appendix	Contents
3	Viewing and Printing Report Output	Describes the HotScreen facility for viewing report output.
4	Sorting Tabular Reports	Describes how to display report information grouped in a particular order by sorting.
5	Selecting Records for Your Report	Describes how to use and specify selection criteria to display only the field values that meet your needs.
6	Creating Temporary Fields	Describes how to use the DEFINE and COMPUTE commands to create temporary fields.
7	Including Totals and Subtotals	Describes how to use subtotals and grand totals to summarize numeric information and aid in interpreting detailed information in a report.
8	Using Expressions	Describes how to combine operators, field names, and constants in an expression to derive new values.
9	Customizing Tabular Reports	Describes how to override the default report formats to meet your individual presentation needs.
10	Saving and Reusing Your Report Output	Describes how to save report data to files for reuse in different respects.
11	Styling Reports	Describes how to visually style your reports with StyleSheets, used to control report output to be printed on a PostScript printer.
12	Cascading Style Sheets	Describes how Cascading Style Sheets (CSS) provide a standardized method for styling HTML documents. To use an existing Cascading Style Sheet, you can link it to your report and, optionally, apply additional CSS classes to specific report components.
13	Working With Styled Output Formats	Describes features of and techniques for working with reports in HTML, PDF, PostScript, Excel 2000, or Excel 97 format.
14	Advanced StyleSheet Features	Describes advanced StyleSheet features such as positioning and aligning elements.

	Chapter/Appendix	Contents
15	Handling Records With Missing Field Values	Describes how missing data affects report results and how to treat and represent it.
16	Joining Data Sources	Describes how to join two or more related data sources to create a larger integrated data structure from which you can report.
17	Merging Data Sources	Describes how to merge and concatenate two or more data sources into a new permanent data source.
18	Improving Report Processing	Describes methods of increasing data retrieval and reporting efficiency.
19	Creating Financial Reports With Financial Modeling Language (FML)	Describes the Financial Modeling Language (FML) used to create and present financially oriented data, using inter-row calculations.
20	Creating a Free-Form Report	Describes how to present data in an unrestricted (non-tabular) format.
21	Creating Graphs: GRAPH	Describes the FOCUS GRAPH facility, which you can use to display data in graph format instead of tabular format.
22	Using SQL to Create Reports	Describes how to use SQL to retrieve and analyze FOCUS and DBMS data.
A	Master Files and Diagrams	Contains Master Files and diagrams of sample data sources used in the documentation examples.
B	Error Messages	Describes how to obtain information about error messages.
C	Table Syntax Summary	Summarizes TABLE commands and options.
D	Writing User-Coded Programs to Create HOLD Files	Describes how to write programs that get records retrieved by FOCUS so you can write them to files in a custom format.

Documentation Conventions

The following table lists and describes the conventions that apply in this manual.

Convention	Description
THIS TYPEFACE or this typeface	Denotes syntax that you must enter exactly as shown.
<i>this typeface</i>	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
<u>underscore</u>	Indicates a default setting.
<i>this typeface</i>	Represents a placeholder (or variable), a cross-reference, or an important term. It may also indicate a button, menu item, or dialog box option you can click or select.
this typeface	Highlights a file name or command.
Key + Key	Indicates keys that you must press simultaneously.
{ }	Indicates two or three choices; type one of them, not the braces.
[]	Indicates a group of optional parameters. None are required, but you may select one of them. Type only the parameter in the brackets, not the brackets.
	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis points (...).
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

To view a current listing of our publications and to place an order, visit our Technical Documentation Library, <http://documentation.informationbuilders.com>. You can also contact the Publications Order Department at (800) 969-4636.

Customer Support

Do you have questions about FOCUS?

Call Information Builders Customer Support Services (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 questions. Information Builders consultants can also give you general guidance regarding product capabilities and documentation. Be prepared to provide your six-digit site code (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>. You can connect 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 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 effectively, be prepared to provide the following information when you call:

- ❑ Your six-digit site code (xxxx.xx).
- ❑ The stored procedure (preferably with line numbers) .
- ❑ The Master File and Access 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
- ? TSO DDNAME or CMS QFI
- 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 return 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, your security system, communications protocol, or front-end software 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

In an effort to produce effective documentation, the Documentation Services staff welcomes your opinions regarding this manual. Please use the Reader Comments form at the end of this manual to communicate suggestions for improving this publication or to alert us to corrections. You can also use the Documentation Feedback form on our Web site, <http://documentation.informationbuilders.com/feedback.asp>.

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.

1 Creating Tabular Reports

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, 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.

Topics:

- ❑ Requirements for Creating a Report
- ❑ Creating a Report Request
- ❑ Developing Your Report Request
- ❑ Including Display Fields in a Report Request
- ❑ Referring to Fields in a Report Request

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, FOCUS must be able to locate the data source.

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, which shows 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

In this section:

- Beginning a Report Request
- Requesting Help When Issuing a Report Request
- Completing a Report Request
- Selecting a Report Output Destination

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

How to:

- Begin 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, if you want your report to be sorted, you need only include a sorting phrase.

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 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 display 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 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, SAVE, or SAVB command. For details, see [Saving and Reusing Your Report Output](#) on page 421.

Developing Your Report Request

The only requirement for reporting is identifying a data source. Beyond that, the structure of a report request is very flexible; that is, you only need to include the report elements you want. For example, you only need to include sorting instructions if you want your report to be sorted, or selection criteria if you want to report on a subset of your data.

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.

The following are the most frequently used options for structuring a report request.

- ❑ **Specifying fields and columns.** Each column in your report represents a field. You can specify which fields you want to display, which fields you want to use to sort the report, which fields you want to use to select records, and which data source fields you want to use in creating temporary fields. Therefore, specifying the fields you want in a report is fundamentally tied to how you want to use those fields in your report.
- ❑ **Displaying data.** You can display data in your report by listing all the records for a field (detailed presentation), or by totaling the records for a field (summary presentation). You can also perform calculations and other operations on fields, such as finding the highest value of a field or calculating the average sum of squares of all the values of a field, and present the results of the operation in your report.

- ❑ **Sorting a report column.** Sorting a report enables you to organize a column's information. FOCUS displays the sort field "the field that controls the sorting order" at the left of the report if you are sorting vertically or at the top if you are sorting horizontally. Sort fields are displayed when their values change. You can also choose not to display sort fields.

You can sort information vertically, down a column, horizontally, across a row; you can also combine vertical sorting and horizontal sorting to create a simple matrix.
- ❑ **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.
- ❑ **Showing subtotals and totals.** You can display column and row totals, grand totals, and section subtotals in your report.
- ❑ **Customizing the presentation.** There are two aspects of a successful report: the information you present, and how it is presented. A report that identifies related groups of information and draws attention to important facts is more effective than one that simply shows columns of data. For example, you can:
 - ❑ Give column titles more meaningful names.
 - ❑ Control the display of columns in your report.
 - ❑ Create headings and footings for different levels of the report "including each sort group, each page, and the entire report, and dynamically control the display of headings and footings based on conditions you set.
 - ❑ Add fonts, colors, grids, and images in a styled report.
 - ❑ Highlight a group of related information and separate it from other groups by inserting blank lines, underlines, and page breaks.
- ❑ **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 temporary fields, deriving their values from real data source fields, and include them in your report. For details, see [Creating Temporary Fields](#) on page 205.
- ❑ **Joining data sources.** You can join two or more data sources to create a larger integrated data structure, from which you can report in a single request. For details, see [Joining Data Sources](#) on page 831.

- ❑ **Storing and reusing the results.** You can store your report data as a data source against which you can make additional queries. This is especially helpful for creating a subset of your data source and for generating two-step reports. You can also format the new data source for use by other data processing tools such as spreadsheets and word processors. For details, see [Saving and Reusing Your Report Output](#) on page 421.

You can run the request as an ad hoc query or save it as a procedure. Saving a report request as a procedure enables you to run or edit it at any time.

Example: Developing a Report Request

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
```

The output is:

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

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. These include 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 64K (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. The total length of fields in the report output is limited to 32K.

Referring to Fields in a Report Request

In this section:

- Referring to an Individual Field
- Referring to Fields Using Qualified Field Names
- Referring to All of the Fields in a Segment
- Displaying a List of Field Names
- Listing Field Names, Aliases, and Format Information

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 40.
- ❑ Refer to fields using qualified field names. For details, see [Referring to Fields Using Qualified Field Names](#) on page 41.
- ❑ 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 43.

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 43 and [Listing Field Names, Aliases, and Format Information](#) on page 44.

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 appears:

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

Referring to Fields Using Qualified Field Names**How to:**

Activate Qualified Field Names

Reference:

Usage Notes for Qualified Field Names

In a request, you can qualify field names with the Master File name and/or the segment name. 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 [Customizing Tabular Reports](#) on page 357.

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 Qualified Field Names

The SET FIELDNAME command enables you to activate qualified field names.

```
SET FIELDNAME = {NEW|OLD|NOTRUNC}
```

where:

NEW

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

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.

NOTRUNC

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

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.

Reference: Usage Notes for 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 check the spelling of a field name, without exiting from the request process. It also shows 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.

2 | Displaying Report Data

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).

Topics:

- ❑ Using Display Commands in a Request
- ❑ Displaying Individual Values
- ❑ Adding Values
- ❑ Counting Values
- ❑ Expanding Byte Precision for COUNT and LIST
- ❑ Maximum Number of Display Fields Supported in a Request
- ❑ Manipulating Display Fields With Prefix Operators
- ❑ Changing the Format of a Report Column

Using Display Commands in a Request

How to:

Use Display Commands in a Request

The 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] [SEG.]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.

SEG.

Displays all fields in a segment (a group of related fields in a Master File). The field name you specify can be any field in the segment.

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 [Maximum Number of Display Fields Supported in a Request](#) on page 59.

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* appears first, followed by the report column for *fieldname2*.

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

AND

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

THE

Is optional and is 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.

Note: The SEG. and * options do not display virtual fields. To print virtual fields, explicitly reference them in the PRINT statement (PRINT * virtual field name). This is true even if the virtual field name is a re-defines of a real field.

Displaying Individual Values

In this section:

Displaying All Fields

Displaying All Fields in a Segment

Displaying the Structure and Retrieval Order of a Multi-Path Data Source

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 49.

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 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 [Optimizing Retrieval Speed for FOCUS Data Sources](#) on page 907, 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 or not the field is a key field, 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. For more information on sorting, see [Sorting Tabular Reports](#) on page 97.

PRINT * or PRINT SEG.* prints only the real fields in the Master File. To print virtual fields, explicitly reference them in the PRINT statement (PRINT * virtual field name). This is true even if the virtual field name is a re-defines of a real field.

For PRINT and LIST syntax, see [How to Use Display Commands in a Request](#) on page 46.

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 following shows the report output.

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 following shows the report output.

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 following shows the report output.

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 All Fields in a Segment

How to:

Display All Fields in a Segment

You can easily display all fields in a segment by adding the prefix "SEG." to any field in the desired segment.

Syntax: How to Display All Fields in a Segment

seg.anyfield

where:

anyfield

Is any field that is in the desired segment.

Example: Displaying All Fields in a Segment

The following request produces a report displaying all of the fields in the segment that contains the QTY_IN_STOCK field.

```
TABLE FILE CENTINV
PRINT SEG.QTY_IN_STOCK
BY PRODNAME NOPRINT
END
```

The following shows the report output.

Product Number:	Product Name:	Quantity In Stock:	Price:	Our Cost:
1028	AR2 35MM Camera 8 X	11499	109.00	79.00
1026	AR3 35MM Camera 10 X	12444	129.00	95.00
1006	Combo Player - 4 Hd VCR + DVD	13527	399.00	289.00
1008	DVD Upgrade Unit for Cent. VCR	199	199.00	139.00
1030	QX Portable CD Player	22000	169.00	99.00
1032	R5 Micro Digital Tape Recorder	1990	89.00	69.00
1036	ZC Digital PDA - Standard	33000	299.00	249.00
1034	ZT Digital PDA - Commercial	21000	499.00	349.00
1024	110 VHS-C Camcorder 20 X	4000	349.00	249.00
1022	120 VHS-C Camcorder 40 X	2300	399.00	259.00
1020	150 8MM Camcorder 20 X	5961	319.00	240.00
1004	2 Hd VCR LCD Menu	43068	179.00	129.00
1018	250 8MM Camcorder 40 X	60073	399.00	320.00
1016	330DX Digital Camera 1024K P	12707	279.00	199.00
1014	340SX Digital Camera 65K P	990	249.00	199.00
1012	650DL Digital Camcorder 150 X	2972	899.00	710.00
1010	750SL Digital Camcorder 300 X	10758	999.00	750.00

Displaying the Structure and Retrieval Order 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 affect your results. You can use the CHECK command PICTURE option to display a diagram of the data source structure defined by the Master File.

You can also display the retrieval order of a data source using the CHECK command PICTURE RETRIEVE option. It should be noted that retrieval is controlled by the minimum referenced subtree. For more information, see *Understanding the Efficiency of the Minimum Referenced Subtree* in the *Describing a Group of Fields* chapter in the *Describing Data* manual.

Example: Displaying the Retrieval Order of a Multi-Path Data Source

To display the retrieval order of the EMPLOYEE data source, which is joined to the JOBFIL and EDUCFIL data sources, issue the following command:

```
CHECK FILE EMPLOYEE PICTURE RETRIEVE
```

The following shows the command output that adds the numbers that display at the top left of each segment, indicating the retrieval order of the segments. 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.

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
```

The following shows a list of the output fields the previous request produces. Due to the size of the report, only the fields for which all instances will be printed are listed here. 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 retrieval order structure of the EMPLOYEE data source, see [Displaying the Retrieval Order of a Multi-Path Data Source](#) on page 51.

Tip: In some environments, the following warning is displayed whenever you use PRINT * with a multi-path data source, to remind you that PRINT * only displays 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 documentation 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 does not add the values; instead, it displays 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 46.

Example: Adding Values

This request adds all the values of the field CURR_SAL:

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

The following shows the output of the request.

```
NUMBER OF RECORDS IN TABLE=      12  LINES=      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 following shows the output of the request.

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

Note that any request for aggregation on all date format fields 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 [Sorting Tabular Reports](#) on page 97.

Counting Values

In this section:

Counting Segment Instances

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 [Sorting Tabular Reports](#) on page 97.

COUNT counts the instances of data contained in a report, not the data values.

For COUNT syntax, see [How to Use Display Commands in a Request](#) on page 46.

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 parameter. For details about the COMPUTE command, see [Creating Temporary Fields](#) on page 205. For information about SET COUNTWIDTH, see the *Developing Applications* manual.

When COUNT is used in a request, the word COUNT is appended to the default column title, unless the column title is changed with an AS phrase.

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 following shows the output of the request.

```
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 following shows the output of the request indicating that of the 12 EMP_IDs in the data source, six are from the MIS department and six are from the PRODUCTION department:

DEPARTMENT	EMP_ID 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 following shows the output of the request.

LAST_NAME COUNT	DEPARTMENT COUNT	JOBCODE 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 parameter, see [Handling Records With Missing Field Values](#) on page 807.

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 following shows the output of the request.

```
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 JOBFIL and EDUCFILE data sources). You can see a picture of the path structure in *Displaying the Structure and Retrieval Order of a Multi-Path Data Source* on page 51.

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**How to:**

Set the Precision for COUNT and LIST

By default, the number of characters that display for counter values retrieved using the COUNT and LIST commands is five. You can increase the number of characters to nine.

For example, if the number of records retrieved for a field exceeds 99,999 (5 bytes), asterisks appear in the report to indicate an overflow condition. You can increase the display to allow as large a count as 999,999,999 (9 bytes) using SET COUNTWIDTH.

Note: You can change the overflow character by issuing the SET OVERFLOWCHAR command.

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 five 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 nine 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 affects the width of a report when COUNTWIDTH is set to ON. Calculating the width of a report now requires an additional four display positions for each COUNT or LIST column.

Maximum Number of Display Fields Supported in a 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. 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 affect 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.

Manipulating Display Fields With Prefix Operators

In this section:

Prefix Operator Basics

Averaging Values of a Field

Averaging the Sum of Squared Fields

Calculating Maximum and Minimum Field Values

Calculating Column and Row Percents

Producing a Direct Percent of a Count

Aggregating and Listing Unique Values

Retrieving First and Last Records

Summing and Counting Values

Ranking Sort Field Values With RNK.

You can use prefix operators to perform calculations directly on the values of fields.

Note: Unless you change a column or ACROSS title with an AS phrase, the prefix operator is automatically added to the title. Without an AS phrase, the column title is constructed using the prefix operator and either the field name or the TITLE attribute in the Master File (if there is one):

- ❑ If there is no TITLE attribute, the field name is used.
- ❑ If there is a TITLE attribute in the Master File, the choice between using the field name or the TITLE attribute depends on the value of the TITLES parameter:
 - ❑ If SET TITLES = ON, the TITLE attribute is used.
 - ❑ If SET TITLES = OFF or NOPREFIX, the field name is used.

For a list of prefix operators and their functions, see [Functions You Can Perform With Prefix Operators](#) on page 62.

Prefix Operator Basics

How to:

Use Prefix Operators

Reference:

Usage Notes for Prefix Operators

Functions You Can Perform With Prefix Operators

This topic describes basic syntax and notes for using prefix operators.

Syntax: **How to Use Prefix Operators**

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

```
{SUM|COUNT} prefix.fieldname AS 'coltitle'
```

```
{PRINT|COMPUTE} RNK.byfield
```

where:

prefix

Is any prefix operator.

fieldname

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

'*coltitle*'

Is the column title for the report column, enclosed in single quotation marks.

byfield

Is the name of a vertical sort field to be ranked 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 [Sorting Tabular Reports](#) on page 97.
- ❑ The WITHIN phrase is very useful when using prefixes.
- ❑ 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

The following table lists prefix operators and describes the function of each.

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. The data type of the result is always Integer.
CNT.DST.	Counts the number of distinct values within a field.
CT.	Produces a cumulative total of the specified field. This operator only applies when used in subfootings.
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 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 percentage based on the number of instances found. The format of the result is always F6.2 and cannot be reformatted.
RNK.	Ranks the instances of a BY sort field in the request. Can be used in PRINT commands, COMPUTE commands, and IF or WHERE TOTAL tests.

Prefix	Function
RPCT.	Computes a field percentage based on the total values for the field across a row.
ST.	Produces a subtotal value of the specified field at a sort break in the report. This operator only applies when used in subfootings.
SUM.	Sums the field values.
TOT.	Totals the field values 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 following shows the output of the request.

```

                AVE
DEPARTMENT    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 (shown as a formula in the following image).

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

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 following shows the output of the request.

CITY	DELIVER_AMT	ASQ DELIVER_AMT
NEW YORK	300	980
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 following shows the output of the request.

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 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 for the total value of the row. The percentage values have the same format as the original field.

Example: Calculating Column Percents

To calculate each employee 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 always 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:

PROD_CODE	CITY		NEWARK		STAMFORD	
	NEW YORK					
	UNIT_SOLD	RPCT	UNIT_SOLD	RPCT	UNIT_SOLD	RPCT
B10	30	29	13	12	60	58
B12	.	.	29	42	40	57
B17	20	40	.	.	29	59
B20	15	37
C13	25	100
C17	12	100
C7	45	52
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 may be truncated and the total percentage may be 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 each row's count within the total number of instances. 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:

DEPARTMENT	PCT.CNT EMP_ID
MIS	50.00
PRODUCTION	50.00

Aggregating and Listing Unique Values

How to:

Use the Distinct Operator

Reference:

Distinct Operator Limitations

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 31 (the current limit for BY sort fields with PRINT).

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

- ❑ If you reformat a column created using COUNT DST. or the CNT.DST operator, you must reformat it to an integer (I) data type. If you specify another data type, the following error occurs:

```
(FOC950) INVALID REFORMAT OPTION WITH COUNT OR CNT.
```

- ❑ 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 [Improving Report Processing](#) on page 903.

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:

```
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:

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:

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 with one display command using the prefix operators `CNT.`, `SUM.`, and `TOT.` Just like the `COUNT` command, `CNT.` counts the occurrences of the field it prefixes. Just like the `SUM` command, `SUM.` sums the values of the field it prefixes. `TOT.` sums the values of the field it prefixes when used in a heading (including 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 Code	Unit Price
COUNT	
-----	-----
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 Code	Unit Price
----- 10	----- 660.00

Example: Summing 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

Ranking Sort Field Values With RNK.

How to:

Calculate Ranks Using the RNK. Prefix Operator

RANKED BY *fieldname*, when used in a sort phrase in a TABLE request, not only sorts the data by the specified field, but assigns a RANK value to the instances. The RNK. prefix operator also calculates the rank while allowing the RANK value to be printed anywhere on the page. You use this operator by specifying RNK.*fieldname*, where *fieldname* is a BY field in the request.

The ranking process occurs after selecting and sorting records. Therefore, the RNK. operator cannot be used in a WHERE or IF selection test or in a virtual (DEFINE) field. However, RNK.*fieldname* can be used in a WHERE TOTAL or IF TOTAL test or in a calculated (COMPUTE) value. You can change the default column title for the rank field using an AS phrase.

You can apply the RNK. operator to multiple sort fields, in which case the rank for each BY field is calculated within its higher level BY field.

Syntax: **How to Calculate Ranks Using the RNK. Prefix Operator**

In a PRINT command, COMPUTE expression, or IF/WHERE TOTAL expression :

```
RNK.field ...
```

where:

field

Is a vertical (BY) sort field in the request.

Example: Ranking Within Sort Groups

The following request ranks years of service within department and ranks salary within years of service and department. Note that years of service depends on the value of TODAY. The output for this example was valid when run in September, 2006:

```

DEFINE FILE EMPDATA
  TODAY/YYMD = &YYMD;
  YRS_SERVICE/I9 = DATEDIF(HIREDATE,TODAY,'Y');
END
TABLE FILE EMPDATA
PRINT SALARY
  RNK.YRS_SERVICE AS 'RANKING,BY,SERVICE'
  RNK.SALARY AS 'SALARY,RANK'
  BY DEPT
  BY HIGHEST YRS_SERVICE
  BY HIGHEST SALARY NOPRINT
WHERE DEPT EQ 'MARKETING' OR 'SALES'
ON TABLE SET PAGE NOPAGE
END
    
```

The output is:

DEPT	YRS_SERVICE	SALARY	RANKING BY SERVICE	SALARY RANK
MARKETING	17	\$55,500.00	1	1
		\$55,500.00	1	1
	16	\$62,500.00	2	1
		\$62,500.00	2	1
		\$62,500.00	2	1
		\$58,800.00	2	2
		\$52,000.00	2	3
		\$35,200.00	2	4
		\$32,300.00	2	5
		\$50,500.00	3	1
\$43,400.00	3	2		
SALES	17	\$115,000.00	1	1
		\$54,100.00	1	2
	16	\$70,000.00	2	1
		\$43,000.00	2	2
	15	\$43,600.00	3	1
		\$39,000.00	3	2
15	\$30,500.00	3	3	

Example: Using RNK. in a WHERE TOTAL Test

The following request displays only those rows in the highest two salary ranks within the years of service category. Note that years of service depends on the value of TODAY. The output for this example was valid when run in September, 2006:

```
DEFINE FILE EMPDATA
  TODAY/YYMD = &YYMD;
  YRS_SERVICE/I9 = DATEDIF(HIREDATE, TODAY, 'Y' );
END
TABLE FILE EMPDATA
PRINT LASTNAME FIRSTNAME RNK.SALARY
BY HIGHEST YRS_SERVICE BY HIGHEST SALARY
WHERE TOTAL RNK.SALARY LE 2
END
```

The output is:

YRS_SERVICE	SALARY	LASTNAME	FIRSTNAME	RANK SALARY
17	\$115,000.00	LASTRA	KAREN	1
	\$80,500.00	NOZAWA	JIM	2
16	\$83,000.00	SANCHEZ	EVELYN	1
	\$70,000.00	CASSANOVA	LOIS	2
15	\$62,500.00	HIRSCHMAN	ROSE	1
		WANG	JOHN	1
	\$50,500.00	LEWIS	CASSANDRA	2

Example: Using RNK. in a COMPUTE Command

The following request sets a flag to Y for records in which the salary rank within department is less than or equal to 5 and the rank of years of service within salary and department is less than or equal to 6. Otherwise, the flag has the value N. Note that the years of service depends on the value of TODAY. The output for this example was valid when run in September, 2006:

```
DEFINE FILE EMPDATA
  TODAY/YYMD = &YYMD;
  YRS_SERVICE/I9 = DATEDIF(HIREDATE, TODAY, 'Y' );
END
PRINT RNK.SALARY RNK.YRS_SERVICE
COMPUTE FLAG/A1 = IF RNK.SALARY LE 5 AND RNK.YRS_SERVICE LE 6
  THEN 'Y' ELSE 'N';
BY DEPT BY SALARY BY YRS_SERVICE
WHERE DEPT EQ 'MARKETING' OR 'SALES'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

DEPT	SALARY	YRS_SERVICE	RANK SALARY	RANK YRS_SERVICE	FLAG
MARKETING	\$32,300.00	16	1		1 Y
	\$35,200.00	16	2		1 Y
	\$43,400.00	15	3		1 Y
	\$50,500.00	15	4		1 Y
	\$52,000.00	16	5		1 Y
	\$55,500.00	17	6		1 N
			6		1 N
	\$58,800.00	16	7		1 N
	\$62,500.00	16	8		1 N
			8		1 N
			8		1 N
SALES	\$30,500.00	15	1		1 Y
	\$39,000.00	15	2		1 Y
	\$43,000.00	16	3		1 Y
	\$43,600.00	15	4		1 Y
	\$54,100.00	17	5		1 Y
	\$70,000.00	16	6		1 N
	\$115,000.00	17	7		1 N

Changing the Format of a Report Column

In this section:

Determining the Width of a Report Column

Reference:

Usage Notes for Changing Column Format

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 [Customizing Tabular Reports](#) on page 357.

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.
- ❑ If you create an extract file from the report, that is, a HOLD, PCHOLD, SAVE, or SAVB file, the extract file contains fields for both the original format and the redefined format, unless HOLDLIST=PRINTONLY. Extract files are described in [Saving and Reusing Your Report Output](#) on page 421.
- ❑ 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 is 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.

Determining the Width of a Report Column

In this section:

Controlling Missing Values for a Reformatted Field

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 needs 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 [Customizing Tabular Reports](#) on page 357.

Controlling Missing Values for a Reformatted Field

When a field is reformatted in a request (for example, SUM *field/format*), an internal COMPUTE field is created to contain the reformatted field value and display on the report output. If the original field has a missing value, that missing value can be propagated to the internal field by setting the COMPMISS parameter ON. If the missing value is not propagated to the internal field, it displays a zero (if it is numeric) or a blank (if it is alphanumeric). If the missing value is propagated to the internal field, it displays the missing data symbol on the report output.

Syntax: How to Control Missing Values in Reformatted Fields

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

where:

ON

Propagates a missing value to a reformatted field. ON is the default value.

OFF

Displays a blank or zero for a reformatted field.

Example: Controlling Missing Values in Reformatted Fields

The following procedure prints the RETURNS field from the SALES data source for store 14Z. With COMPMISS OFF, the missing values display as zeros in the column for the reformatted field value. (**Note:** Before trying this example, you must make sure that the SALEMISS procedure, which adds missing values to the SALES data source, has been run.)

```
SET COMPMISS = OFF
TABLE FILE SALES
PRINT RETURNS RETURNS/D12.2 AS 'REFORMATTED,RETURNS'
BY STORE_CODE
WHERE STORE_CODE EQ '14Z'
END
```

The output is:

STORE_CODE	RETURNS	REFORMATTED RETURNS
-----	-----	-----
14Z	2	2.00
	2	2.00
	0	.00
	.	.00
	4	4.00
	0	.00
	3	3.00
	4	4.00
	.	.00
	4	4.00

With COMPMISS ON, the column for the reformatted version of RETURNS displays the missing data symbol when a value is missing:

```
SET COMPMISS = ON
TABLE FILE SALES
PRINT RETURNS RETURNS/D12.2 AS 'REFORMATTED,RETURNS'
BY STORE_CODE
WHERE STORE_CODE EQ '14Z'
END
```

The output is:

STORE_CODE	RETURNS	REFORMATTED RETURNS
-----	-----	-----
14Z	2	2.00
	2	2.00
	0	.00
	.	.
	4	4.00
	0	.00
	3	3.00
	4	4.00
	.	.
	4	4.00

Reference: Usage Notes for SET COMPMISS

- ❑ If you create a HOLD file with COMPMISS ON, the HOLD Master File for the reformatted field indicates MISSING = ON (as does the original field). With COMPMISS = OFF, the reformatted field does NOT have MISSING = ON in the generated Master File.
- ❑ The COMPMISS parameter cannot be set in an ON TABLE command.

3 | Viewing and Printing Report Output

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.

Topics:

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

Displaying Reports in Hot Screen

In this section:

Using PRINTPLUS

Accessing Help Information

How to:

Activate 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. 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 the following query at the FOCUS command prompt. The value of the SCREEN parameter should be ON:

```
? SET SCREEN
```

If you are using a full-screen terminal, you can activate Hot Screen by issuing the following command at the FOCUS command prompt:

```
SET SCREEN=ON
```

This is the default setting for full-screen terminals.

Other acceptable values for SCREEN 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

How to:

Use 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 to use the following TABLE capabilities:

- ❑ 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, WebFOCUS 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.
- ❑ Data displays correctly in subfoots when IF/WHERE TOTAL or BY HIGHEST is used.
- ❑ BY field actions are linked with BY field options so they appear on the same page. The footing no longer splits on two pages.
- ❑ Footings and Subfoots always appear on a page with at least one data item, and will never split between two pages.
- ❑ Printing beyond the length of the page no longer occurs.
- ❑ Splitting of fields linked by OVER onto separate pages no longer occurs.
- ❑ There is no reserved space for conditional output. The output page is fully used.
- ❑ The order of sort fields is no longer relevant.

Note: PRINTPLUS is not supported for StyleSheets. A warning message is generated in this case.

Syntax: How to Use PRINTPLUS

Issue the command

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

Example: Using PRINTPLUS With SUBFOOT and FOOTING

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

```
SET PRINTPLUS = ON
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			

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

In this section:

- Scrolling Forward
- Scrolling Backward
- Scrolling Horizontally
- Scrolling From Fixed Columns (Fencing)
- Scrolling Report Headings
- Saving Selected Data
- Locating Character Strings
- Repeating Commands
- Redisplaying Reports
- Previewing Your Report
- Displaying BY Fields With Panels
- Scrolling by Columns of BY Fields
- The SET COLUMNS Command

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, save data, locate character strings, repeat commands, redisplay a report, preview a report, and display BY fields with panels.

Scrolling Forward

To scroll forward in a report 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 returns to the FOCUS command line.

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

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

Command	Description
<code>TOP</code>	Scrolls the display directly back to the first page of the report.
<code>UP n</code>	Scrolls the display back the number of pages you specify.
<code>BACK n</code>	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.

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. The 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 do not scroll. Note that fencing is not supported while BYPANEL is on. To determine the setting of BYSCROLL, enter ? SET 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 saves the text from that start character to the end of the line in a file with the file name SAVE. See [Saving and Reusing Your Report Output](#) on page 421 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 prompts for the string:

`ENTER STRING TO LOCATE /`

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

FOCUS searches from the current position forward. When it locates the string, 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 remains 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 with 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 are supported, except for alpha (A), text (TX), dates, and fields with date edit options.

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 FML.
- ❑ When reformatting a packed field, you may not change the number of places after the decimal point. 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 to 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

Reference:

BYPANEL Conditions

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 or within the request (using the ON TABLE phrase):

Value	Description
ON	Displays all BY fields specified in the report on each panel, and prevents column splitting.
<i>n</i>	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. This prevents column splitting. 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.

Value	Description
0	Zero displays BY fields only on the first panel. This prevents column splitting.
<u>OFF</u>	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
```

The output is:

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 is ignored.

- ❑ In OFFLINE reports, the SET PANEL command is respected when used with SET BYPANEL. If you choose to override the report width, define a panel large enough to enable the BYPANEL feature using the SET PANEL command. 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 is repeated from the beginning on the panels which follow.
- ❑ 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 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

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 as 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 automatically prompts 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

In this section:

The OFFLINE Command
Printing Reports in Hot Screen

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 directs all offline reports to the default output spool file. This file is assigned automatically when you enter FOCUS, 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 simply by entering the following two commands at the FOCUS command prompt:

```
OFFLINE  
RETYPE
```

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 z/OS. The FILEDEF and DYNAM commands are described in the *Overview and Operating Environments* manual.

You can reroute report output to your screen by issuing the following command at the FOCUS command prompt:

ONLINE

You can reroute it only for the current request by including an ON TABLE SET PRINT ONLINE command in the request. Be sure also to issue the following command to close any current spool file and enable you to allocate new ones:

OFFLINE CLOSE

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 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.

4 | **Sorting Tabular Reports**

Sorting enables you to group or organize report information vertically and horizontally, in rows and columns, and specify a desired 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 appear only when their values change.

Topics:

- ❑ [Sorting Tabular Reports Overview](#)
- ❑ [Sorting Rows](#)
- ❑ [Sorting Columns](#)
- ❑ [Manipulating Display Field Values in a Sort Group](#)
- ❑ [Creating a Matrix Report](#)
- ❑ [Specifying the Sort Order](#)
- ❑ [Ranking Sort Field Values](#)
- ❑ [Grouping Numeric Data Into Ranges](#)
- ❑ [Restricting Sort Field Values by Highest/Lowest Rank](#)
- ❑ [Sorting and Aggregating Report Columns](#)
- ❑ [Hiding Sort Values](#)
- ❑ [Sorting With Multiple Display Commands](#)
- ❑ [Improving Efficiency With External Sorts](#)

Sorting Tabular Reports Overview

Reference:

Sorting and Displaying Data

You sort a report using vertical (BY) and horizontal (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 phrases used in the same report create rows and columns, producing a grid or matrix.

Additional sorting options include:

- ❑ Sorting from low to high values or from high to low values, and defining your own sorting sequence.
- ❑ Leaving the value of the sort field out of the report.
- ❑ Grouping numeric data into tiles such as percentiles or deciles.
- ❑ Aggregating and sorting numeric columns simultaneously.
- ❑ Grouping numeric data into ranges.
- ❑ Ranking data, and selecting data based on rank.

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 139.

For details on all display commands, see [Displaying Report Data](#) on page 45.

Sorting Rows

In this section:

Displaying All Vertical (BY) Sort Field Values

Using Multiple Vertical (BY) Sort Fields

Displaying a Row for Data Excluded by a Sort Phrase

How to:

Sort by Rows

Reference:

Usage Notes for 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).

Sort fields appear when their value changes. However, you can display every sort value using the BYDISPLAY parameter. For an example, see [Displaying All Vertical \(BY\) Sort Field Values](#) on page 101.

Syntax: **How to Sort by Rows**

`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.
- ❑ The default sort sequence is low-to-high, with the following variations for different operating systems. In z/OS and VM the sequence is a-z, A-Z, 0-9 for alphanumeric fields; 0-9 for numeric fields. In UNIX and Windows 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 116.
- ❑ You cannot use text fields as sort fields. Text fields are those described in the Master File with a FORMAT value of TX.

- ❑ You can use a temporary field created by a DEFINE command, or by the DEFINE attribute in a Master File, as a sort field. However, you cannot use a temporary field created by a COMPUTE command as a sort field. 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 [Saving and Reusing Your Report Output](#) on page 421).
- ❑ 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.
- ❑ 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. You can populate every vertical sort column cell with a value, even if the value is repeating, using the SET BYDISPLAY parameter. For details, see [Displaying All Vertical \(BY\) Sort Field Values](#) on page 101.

Example: Sorting Rows With BY

The following illustrates how to display all employee IDs by department.

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

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

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

Displaying All Vertical (BY) Sort Field Values

How to:

Display All Vertical (BY) Sort Field Values

Within a vertical sort group, the sort field value displays only on the first line of the rows for its sort group, and on the first line of a page. However, using the SET BYDISPLAY command, you can display the appropriate BY field on every row of a report produced in a styled output format.

Although SET BYDISPLAY is supported for all styled output formats, it is especially important for making report output more usable by Excel, which cannot sort columns properly when they have blank values in some rows.

This feature enables you to avoid specifying the sort field twice, once as a display field and once for sorting (with the NOPRINT option). For example:

```
PRINT FIRST_NAME LAST_NAME
BY FIRST_NAME NOPRINT
```

Syntax: How to Display All Vertical (BY) Sort Field Values

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

or

```
ON TABLE SET BYDISPLAY {OFF|ON}
```

where:

OFF

Displays a BY field value only on the first line of the report output for the sort group and on the first line of a page. OFF is the default value.

ON

Displays the associated BY field value on every line of report output produced in a styled format.

Example: Displaying All Vertical (BY) Sort Field Values

The following illustrates how you can display every instance of a vertical (BY) sort field value in a styled report using SET BYDISPLAY.

```
SET BYDISPLAY = ON
TABLE FILE CENTHR
PRINT LNAME
BY FNAME
WHERE FNAME EQ 'CAROLYN' OR 'DAVID' ON
TABLE HOLD FORMAT EXL2K
END
```

The output is:

	A	B	C	D
	First Name	Last Name	Plant Location	Pay Level
1	CAROLYN	MILLER	LA	5
2	CAROLYN	STRICKER	SEA	4
3	DAVID	JAMES	ORL	3
4	DAVID	KERNER	SEA	7
5	DAVID	MONTGOMERY	ORL	5
6	DAVID	RODRIGUEZ	SEA	6
7	DAVID	TEPPER	BOS	2
8	DAVID	HAZARD	BOS	6
9	DAVID	SEGAL	BOS	4
10	DAVID	KNUDSEN	LA	5
11	DAVID	PARADISE	LA	5
12	DAVID	HENSLEE	DAL	5

Using Multiple Vertical (BY) Sort Fields

You can organize information in a report by 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 Vertical (BY) Sort Fields

The following request uses multiple vertical (BY) sort fields.

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

The output is:

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

Displaying a Row for Data Excluded by a Sort Phrase**How to:**

Display Data Excluded by a Sort Phrase

Reference:

Usage Notes for PLUS OTHERS

In a sort phrase, you can restrict the number of sort values displayed. With the PLUS OTHERS phrase, you can aggregate all other values to a separate group and display this group as an additional report row.

Syntax: How to Display Data Excluded by a Sort Phrase

```
[RANKED] BY {HIGHEST|LOWEST|TOP|BOTTOM} nsrtfield [AS 'text']
           [PLUS OTHERS AS 'othertext']
           [IN-GROUPS-OF m1 [TOP n2]]
           [IN-RANGES-OF m3 [TOP n4]]
```

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). BOTTOM is a synonym for LOWEST.

HIGHEST

Sorts in descending order, beginning with the highest value and continuing to the lowest value. TOP is a synonym for HIGHEST.

n

Specifies that only *n* sort field values are included in the report.

srtfield

Is the name of the sort field.

text

Is the text to be used as the column heading for the sort field values.

othertext

Is the text to be used as the row title for the "others" grouping. This AS phrase must be the AS phrase immediately following the PLUS OTHERS phrase.

m1

Is the incremental value between sort field groups.

n2

Is an optional number that defines the highest group label to be included in the report.

m3

Is an integer greater than zero indicating the range by which sort field values are grouped.

n4

Is an optional number that defines the highest range label to be included in the report. The range is extended to include all data values higher than this value.

Reference: Usage Notes for PLUS OTHERS

- ❑ Alphanumeric group keys are not supported.
- ❑ Only one PLUS OTHERS phrase is supported in a request.
- ❑ In a request with multiple display commands, the BY field that has the PLUS OTHERS phrase does not have to be the last BY field in the request.
- ❑ The BY ROWS OVER, TILES, ACROSS, and BY TOTAL phrases are not supported with PLUS OTHERS.
- ❑ PLUS OTHERS is not supported in a MATCH FILE request. However, MORE in a TABLE request is supported.
- ❑ HOLD is supported for formats PDF, PS, HTML, DOC, and WP.

Example: Displaying a Row Representing Sort Field Values Excluded by a Sort Phrase

The following request displays the top two ED_HRS values and aggregates the values not included in a row labeled Others:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL LAST_NAME
  BY HIGHEST 2 ED_HRS
  PLUS OTHERS AS 'Others'
END
```

The output is:

ED_HRS	CURR_SAL	LAST_NAME
-----	-----	-----
75.00	\$21,780.00	BLACKWOOD
50.00	\$18,480.00	JONES
	\$16,100.00	MCKNIGHT
Others	\$165,924.00	

Example: Displaying a Row Representing Data Not Included in Any Sort Field Grouping

The following request sorts by highest 2 ED_HRS and groups the sort field values by increments of 25 ED_HRS. Values that fall below the lowest group label are included in the Others category. All values above the top group label are included in the top group:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL LAST_NAME
  BY HIGHEST 2 ED_HRS
  PLUS OTHERS AS 'Others'
IN-GROUPS-OF 25 TOP 50
END
```

The output is:

ED_HRS	CURR_SAL	LAST_NAME
-----	-----	-----
50.00	\$18,480.00	JONES
	\$21,780.00	BLACKWOOD
	\$16,100.00	MCKNIGHT
25.00	\$11,000.00	STEVENS
	\$13,200.00	SMITH
	\$26,862.00	IRVING
	\$9,000.00	GREENSPAN
	\$27,062.00	CROSS
Others	\$78,800.00	

If the BY HIGHEST phrase is changed to BY LOWEST, all values above the top grouping (50 ED_HRS and above) are included in the Others category:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL LAST_NAME
  BY LOWEST 2 ED_HRS
  PLUS OTHERS AS 'Others'
IN-GROUPS-OF 25 TOP 50
END
```

The output is:

ED_HRS	CURR_SAL	LAST_NAME
.00	\$9,500.00	SMITH
	\$29,700.00	BANNING
	\$21,120.00	ROMANS
	\$18,480.00	MCCOY
25.00	\$11,000.00	STEVENS
	\$13,200.00	SMITH
	\$26,862.00	IRVING
	\$9,000.00	GREENSPAN
	\$27,062.00	CROSS
Others	\$56,360.00	

Sorting Columns

In this section:

- Controlling Underlines for ACROSS Objects
- Using Multiple Horizontal (ACROSS) Sort Fields
- Collapsing PRINT With ACROSS

How to:

- Sort Columns

Reference:

- Usage Notes for 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 generate up to 255 columns of data. 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. In general, if a horizontal (ACROSS) sort field contains many data values, you may exceed the allowed width for reports, or create a report that is difficult to read. For details, see [Displaying Report Data](#) on page 45.

You can produce column totals or summaries for ACROSS sort field values using ACROSS-TOTAL, SUBTOTAL, SUB-TOTAL, RECOMPUTE, and SUMMARIZE. For details, see [Including Totals and Subtotals](#) on page 269.

Syntax: How to Sort Columns

`ACROSS sortfield`

where:

`sortfield`

Is the name of the sort field.

Reference: Usage Notes for Sorting Columns

- ❑ You cannot use text fields as sort fields. Text fields are those described in the Master File with a FORMAT value of TX.
- ❑ You can use a temporary field created by a DEFINE command, or by the DEFINE attribute in a Master File, as a sort field. However, you cannot use a temporary field created by a COMPUTE command as a sort field. 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 [Saving and Reusing Your Report Output](#) on page 421.
- ❑ For an ACROSS phrase, the SET SPACES parameter controls the distance between ACROSS sets. For more information, see [Customizing Tabular Reports](#) on page 357.
- ❑ 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.
- ❑ 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
```

prints MIS once, followed by six employee names.

Example: Sorting Columns With ACROSS

The following illustrates how to show the total salary outlay for each department. This request is sorted horizontally with an ACROSS phrase.

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

The output is:

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

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

Controlling Underlines for ACROSS Objects

How to:
Control Underlining for ACROSS Objects

The SET ACROSSLINE command allows users to turn off/on optional underlining in reports to highlight ACROSS objects. The feature is only available for Hotscreen reports and report output formats WP, HTML, and PDF.

Syntax: How to Control Underlining for ACROSS Objects

Issue the following command in any supported profile, or in a FOCEXEC, or at the command prompt:

```
SET ACROSSLINE= (ON|OFF|SKIP)
```

where:

ON
Underlines ACROSS objects in report headings with a dashed line. ON is the default value.

OFF
Replaces the underline with a blank line.

SKIP
Specifies no underline and no blank line.

Example: Underlining ACROSS Objects With a Dashed Line (SET ACROSSLINE=ON)

```
SET ACROSSLINE=ON
TABLE FILE GGSALES
SUM UNITS BY PRODUCT
ACROSS REGION
END
```

The output is:

Product	Region			
	Midwest	Northeast	Southeast	West
Biscotti	86105	145242	119594	70436
Cappuccino	.	44785	73264	71168
Coffee Grinder	50393	40977	47083	48081
Coffee Pot	47156	46185	49922	47432
Croissant	139182	137394	156456	197022
Espresso	101154	68127	68030	71675
Latte	231623	222866	209654	213920
Mug	86718	91497	88474	93881
Scone	116127	70732	73779	72776
Thermos	46587	48870	48976	45648

Example: Removing Underlines for ACROSS Objects (SET ACROSSLINE=SKIP)

```
SET ACROSSLINE=SKIP
TABLE FILE GGSALES
SUM UNITS BY PRODUCT
ACROSS REGION
END
```

The output is:

Product	Region			
	Midwest	Northeast	Southeast	West
Biscotti	86105	145242	119594	70436
Cappuccino	.	44785	73264	71168
Coffee Grinder	50393	40977	47083	48081
Coffee Pot	47156	46185	49922	47432
Croissant	139182	137394	156456	197022
Espresso	101154	68127	68030	71675
Latte	231623	222866	209654	213920
Mug	86718	91497	88474	93881
Scone	116127	70732	73779	72776
Thermos	46587	48870	48976	45648

Example: Replacing the Underline With a Blank Line (SET ACROSSLINE=OFF)

```
SET ACROSSLINE=OFF
TABLE FILE GGSALES
SUM UNITS BY PRODUCT
ACROSS REGION
END
```

Turning ACROSSLINE=OFF replaces the (default) dashed line with an extra blank line between the report heading and the detail lines:

Product	Region			
	Midwest	Northeast	Southeast	West
Biscotti	86105	145242	119594	70436
Cappuccino	.	44785	73264	71168
Coffee Grinder	50393	40977	47083	48081
Coffee Pot	47156	46185	49922	47432
Croissant	139182	137394	156456	197022
Espresso	101154	68127	68030	71675
Latte	231623	222866	209654	213920
Mug	86718	91497	88474	93881
Scone	116127	70732	73779	72776
Thermos	46587	48870	48976	45648

Using Multiple Horizontal (ACROSS) Sort Fields

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 Horizontal (ACROSS) Phrases

The following request sorts the sum of current salaries, first by department and then by job code.

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

The output is:

DEPARTMENT		PRODUCTION	
MIS			
CURR_JOBCODE			
A17	B04	A15	A17
\$27,062.00	\$21,780.00	\$26,862.00	\$29,700.00

Collapsing PRINT With ACROSS

How to:

Compress Report Lines

Reference:

Usage Notes for SET ACROSSPRT

The PRINT command generates a report that has a single line for each record retrieved from the data source after screening out those that fail IF or WHERE tests. When PRINT is used in conjunction with an ACROSS phrase, many of the generated columns may be empty. Those columns display the missing data symbol.

To avoid printing such a sparse report, you can use the SET ACROSSPRT command to compress the lines in the report. The number of lines is reduced within each sort group by swapping non-missing values from lower lines with missing values from higher lines, and then eliminating any lines whose columns all have missing values.

Because data may be moved to different report lines, row-based calculations such as ROW-TOTAL and ACROSS-TOTAL in a compressed report are different from those in a non-compressed report. Column calculations are not affected by compressing the report lines.

Syntax: How to Compress Report Lines

```
SET ACROSSPRT = {NORMAL | COMPRESS}
```

```
ON TABLE SET ACROSSPRT {NORMAL | COMPRESS}
```

where:

NORMAL

Does not compress report lines. NORMAL is the default value.

COMPRESS

Compresses report lines by promoting data values up to replace missing values within a sort group.

Reference: Usage Notes for SET ACROSSPRT

- ❑ Compression applies only to ACROSS fields, including ACROSS ... COLUMNS. It has no effect on BY fields.
- ❑ The only data values that are subject to compression are true missing values. If the value of the stored data is either 0 or blank and the metadata indicates that MISSING is ON, that value is not subject to compression.

Example: Compressing Report Output With SET ACROSSPRT

The following request against the GGSALES data source prints unit sales by product across region:

```
TABLE FILE GGSALES
PRINT UNITS/I5
BY PRODUCT
ACROSS REGION
WHERE DATE FROM '19971201' TO '19971231';
WHERE PRODUCT EQ 'Capuccino' OR 'Espresso';
ON TABLE SET ACROSSPRT NORMAL
ON TABLE SET PAGE NOPAGE
END
```

Each line of the report represents one sale in one region, so at most one column in each row has a non-missing value when ACROSSPRT is set to NORMAL:

Product	Region			
	Midwest Unit Sales	Northeast Unit Sales	Southeast Unit Sales	West Unit Sales
Capuccino	.	936	.	.
	.	116	.	.
	.	136	.	.
	.	.	1616	.
	.	.	1118	.
	.	.	774	.
	.	.	.	1696
	.	.	.	1519
	.	.	.	836
Espresso	1333	.	.	.
	280	.	.	.
	139	.	.	.
	.	1363	.	.
	.	634	.	.
	.	406	.	.
	.	.	1028	.
	.	.	1014	.
	.	.	885	.
	.	.	.	1782
	.	.	.	1399
	.	.	.	551

Setting ACROSSPRT to COMPRESS promotes non-missing values up to replace missing values within the same BY group and then eliminates lines consisting of all missing values:

```
TABLE FILE GGSales
PRINT UNITS/I5
BY PRODUCT
ACROSS REGION
WHERE DATE FROM '19971201' TO '19971231';
WHERE PRODUCT EQ 'Capuccino' OR 'Espresso';
ON TABLE SET ACROSSPRT COMPRESS ON TABLE SET PAGE NOPAGE

END
```

The output is:

Product	Region			
	Midwest Unit Sales	Northeast Unit Sales	Southeast Unit Sales	West Unit Sales
Capuccino	.	936	1616	1696
	.	116	1118	1519
	.	136	774	836
Espresso	1333	1363	1028	1782
	280	634	1014	1399
	139	406	885	551

Manipulating Display Field Values in a Sort Group

How to:

Use WITHIN to Manipulate Display Fields

You can use the WITHIN phrase to manipulate a display field 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.

You can use up to 64 fields in a display command when using the WITHIN phrase. The WITHIN phrase requires a BY phrase and/or an ACROSS phrase. A maximum of two WITHIN phrases can be used per display field. 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.

You can also use 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.

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.

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:

STORE_CODE	PROD_CODE	UNITS	PCT SOLD WITHIN TABLE	PCT SOLD WITHIN STORE
-----	-----	-----	-----	-----
K1	B10	13	2	30
	B12	29	4	69
14B	B10	60	9	15
	B12	40	6	10
	B17	29	4	7
	C13	25	3	6
	C7	45	6	11
	D12	27	4	7
	E2	80	12	21
	E3	70	10	18
14Z	B10	30	4	18
	B17	20	3	12
	B20	15	2	9
	C17	12	1	7
	D12	20	3	12
	E1	30	4	18
	E3	35	5	21
77F	B20	25	3	38
	C7	40	6	61

Creating a Matrix Report

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 multiple 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:

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	.

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:

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

Specifying the Sort Order

In this section:

- Specifying Your Own Sort Order

How to:

- Specify the Sort Order

Sort field values are automatically displayed in ascending order, beginning with the lowest value and continuing to the highest. The default sorting sequence varies for operating systems. On z/OS and VM it is a-z, A-Z, 0-9 for alphanumeric fields; 0-9 for numeric fields. On UNIX and Windows 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 specify this same ascending order explicitly by including LOWEST in the sort phrase.

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

The output is:

CURR_SAL	LAST_NAME
\$9,000.00	GREENSPAN
\$9,500.00	SMITH
\$11,000.00	STEVENS
\$13,200.00	SMITH
\$16,100.00	MCKNIGHT
\$18,480.00	JONES
	MCCOY
\$21,120.00	ROMANS
\$21,780.00	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
\$26,862.00	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

How to:

Define Your Own Sort Order

Define Column Sort Sequence

Reference:

Usage Notes for Defining Your Sort Order

Usage Notes for Defining Column Sort Sequence

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

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 those values are included in the report.
2. The order in which 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:

- ❑ The BY ROWS OVER phrase, for defining your own row sort sequence.
- ❑ The ACROSS COLUMNS AND phrase, 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 [Creating Financial Reports With Financial Modeling Language \(FML\)](#) on page 937.

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 only one BY ROWS OVER phrase. BY ROWS OVER is not supported with the FOR phrase. For information about the FOR phrase, see [Creating Financial Reports With Financial Modeling Language \(FML\)](#) on page 937.

Example: Defining Your Row Sort Order

The following illustrates how to sort employees by the banks at which their paychecks are automatically deposited, and how to define your own label in the sorting sequence for the bank field.

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

The output is:

	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.
- ❑ When using a COMPUTE with an ACROSS COLUMNS phrase, the COLUMNS should be specified last:

```
ACROSS acrossfield [AND] COMPUTE compute_expression; COLUMNS values
```

- ❑ Each report request may contain only one BY ROWS OVER phrase.

Example: Defining Column Sort Sequence

The following illustrates how 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.

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

The output is:

BANK_NAME

BEST BANK	STATE	ASSOCIATED	BANK ASSOCIATION
\$29,700.00	\$18,480.00	\$64,742.00	\$27,062.00

Ranking Sort Field Values

How to:

Rank 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.

Note that it is possible for several report rows to have the same rank if they have identical sort field values.

The default column title for RANKED BY is RANK. You can change the title using an AS phrase. The RANK field has format I7. Therefore, the RANK column in a report can be up to seven digits.

You can rank aggregated values using the syntax RANKED BY TOTAL. For details, see [Sorting and Aggregating Report Columns](#) on page 135.

Syntax: How to Rank Sort Field Values

```
RANKED [AS 'name'] BY sortfield
```

where:

sortfield

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

name

Is the new name for the RANK column title.

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. Note that employees Jones and McCoy have the same rank since their current salary is the same.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
RANKED AS 'Sequence' BY CURR_SAL
END
```

The output is:

Sequence	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

Grouping Numeric Data Into Ranges

In this section:

Grouping Numeric Data Into Tiles

How to:

Define Groups of Equal Range

Define Equal Ranges

Define Custom Groups of Data Values

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 several ways of defining groups. You can define groups of:

- ❑ Equal range using the IN-GROUPS-OF phrase.

Each report request can contain a total of five IN-GROUPS-OF phrases plus IN-RANGES-OF phrases. The IN-GROUPS-OF phrase can only be used once per BY field. The first sort field range starts from the lowest value of a multiple of the IN-GROUPS-OF value, and the value displayed is the start point of each range.

- ❑ Equal range using the IN-RANGES-OF phrase.

Each report request can contain a total of five IN-GROUPS-OF phrases plus IN-RANGES-OF phrases. The IN-RANGES-OF phrase can only be used once per BY field. The first sort field range starts from the lowest value of a multiple of the IN-GROUPS-OF value. No message is generated if you specify a range of zero, but the values displayed on the report are unpredictable.

- ❑ Unequal range using the FOR phrase.

- ❑ Tiles. These include percentiles, quartiles, or deciles. For details, see [Grouping Numeric Data Into Tiles](#) on page 127.

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 [Creating Financial Reports With Financial Modeling Language \(FML\)](#) on page 937.

Note: If there is not any data for a group, a row for the group still appears in the report.

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: its format must be I (integer), F (floating-point number), D (decimal number), or P (packed number).

value

Is a positive integer that specifies the range by which sort field values are grouped.

limit

Is an optional number that defines the highest group label to be included in the report.

Example: Defining Groups of Equal Ranges

The following illustrates how to show which employees fall into which salary ranges, and to define the ranges by \$5,000 increments.

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

The output is:

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 Equal Ranges

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

where:

sortfield

Is the name of the sort field. The sort field must be numeric: its format must be I (Integer), F (floating-point), D (double-precision), or P (packed).

value

Is an integer greater than zero indicating the range by which sort field values are grouped.

limit

Is an optional number that defines the highest range label to be included in the report. The range is extended to include all data values higher than this value.

Example: Defining Equal Ranges

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

The output is:

CURR_SAL		LAST_NAME
-----		-----
\$5,000.00 -	\$9,999.99	SMITH GREENSPAN
\$10,000.00 -	\$14,999.99	STEVENS SMITH
\$15,000.00 -	\$19,999.99	JONES MCCOY MCKNIGHT
\$20,000.00 -	\$24,999.99	ROMANS BLACKWOOD
\$25,000.00 -	\$29,999.99	BANNING IRVING CROSS

Syntax: How to Define Custom Groups of Data Values

```
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:

```
          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**How to:**

Group Numeric Data Into Tiles

Reference:

Usage Notes for Tiles

You can group numeric data into any number of tiles (percentiles, deciles, quartiles, etc.) in tabular reports. For example, you can group students' test scores into deciles to determine which students are in the top ten percent of the class, or determine which salesmen are in the top half of all salesmen based on total sales.

Grouping is based on the values in the selected vertical (BY) field, and data is apportioned 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 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 into groups containing an equal number of records 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, while 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 z/OS and VM, values with decimals are rounded to integers; on UNIX and Windows 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 controls the display. If *k* and *m* are both greater than *n* (the number of tiles), *n* is used.

Example: Grouping Data Into Five Tiles

The following illustrates how to group 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
			SCOOBY-DOO-A DOG IN THE RUFF
	26.99	3	BAMBI
		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
	19.98	4	ROBOCOP
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

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
CHILDREN	14.95	1	RAMBO III
	14.98	1	SESAME STREET-BEDTIME STORIES AND SONGS
	19.95	2	ROMPER ROOM-ASK MISS MOLLY
	26.99	3	SMURFS, THE
			SCOOBY-DOO-A DOG IN THE RUFF
			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
CHILDREN	14.95	1	RAMBO III
	14.98	2	SESAME STREET-BEDTIME STORIES AND SONGS
	19.95	3	ROMPER ROOM-ASK MISS MOLLY
			SMURFS, THE
			SCOOBY-DOO-A DOG IN THE RUFF

Reference: Usage Notes for Tiles

- ❑ If a request retrieves data from segments that are descendants 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 output 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 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 sort fields, tiles are supported only at the lowest level and only with the BY LOWEST phrase.
- ❑ Tiles are supported with output files. However, the field used to calculate the tiles propagates three fields to a HOLD file (the actual field value, the tile, and a ranking field) 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**How to:**

Restrict 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. You choose the number of fields to include in the report. For example, you can choose to display only the 10 highest (or lowest) sort field values in your report by using BY HIGHEST (or LOWEST).

You can have up to five sort fields with BY HIGHEST or BY LOWEST.

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 are included in the report. TOP is a synonym for HIGHEST.

LOWEST *n*

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

sortfield

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

Note: 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.

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:

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

Sorting and Aggregating Report Columns

In this section:

Restricting the Number of Columns in a Report

How to:

Sort and Aggregate a Report Column

Using the BY TOTAL phrase, you can apply aggregation and sorting simultaneously to numeric columns in your report in one pass of the data. 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. Records are sorted in either ascending or descending sequence, based on your query. Ascending order is the default.

You can also use the BY TOTAL phrase to sort based on temporary values calculated by the COMPUTE command.

Note: On z/OS 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/COMPUTE name/format=expression;}
```

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. The default order is from lowest to highest.

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 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 WHOLESALPR CNT.WHOLESALPR
BY CATEGORY
BY HIGHEST 2 TOTAL AVE.WHOLESALPR AS 'AVE.WHOLESALPR'
BY RATING
WHERE CATEGORY EQ 'CLASSIC' OR 'FOREIGN' OR 'MUSICALS'
END
```

The output is:

CATEGORY	AVE.WHOLESALPR	RATING	WHOLESALPR	WHOLESALPR 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: Sorting, Aggregating, and Ranking 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 and ranked.

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

The output is:

CATEGORY	RANK	AVE.WHOLESALPR	RATING	WHOLESALPR	WHOLESALPR 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

Example: Sorting and Aggregating Report Columns With COMPUTE

In this example, the monthly salary is calculated using a COMPUTE within a sort field. The two highest monthly salaries are displayed.

```
TABLE FILE EMPLOYEE
SUM SALARY CNT.SALARY
BY DEPARTMENT
BY HIGHEST 2 TOTAL COMPUTE MONTHLY_SALARY/D12.2M=SALARY/12;
AS 'HIGHEST,MONTHLY,SALARIES'
BY CURR_JOBCODE
END
```

The output is:

DEPARTMENT	HIGHEST MONTHLY SALARIES	CURR_JOBCODE	SALARY	SALARY COUNT
-----	-----	-----	-----	-----
MIS	\$4,403.08	A17	\$52,837.00	2
	\$3,019.17	B03	\$36,230.00	2
PRODUCTION	\$4,273.50	A15	\$51,282.00	2
	\$2,591.67	B02	\$31,100.00	2

Restricting the Number of Columns in a Report

The maximum number of report columns created using the ACROSS phrase in a report is 255. The actual number of columns created by ACROSS depends on the number of:

- fields being displayed.
- values for each of these fields.
- ACROSS and ACROSS-TOTAL phrases in the request.

Note: If you exceed the maximum number of ACROSS columns, a message displays.

Other factors that affect the number of ACROSS columns allowed in a report include the sizes of:

- Column titles created using the AS phrase.
- Fields displayed in the report, since the maximum width of a report is 32K bytes.

Row totals, BY columns, SUBHEADs, SUBFOOTs, and fields used in headings and footings do not count in calculating the number of ACROSS columns.

Hiding Sort Values

How to:

Hide Sort Values

When you sort a report, you can omit the sort field value itself from the report by 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, or when you want to sort by a field but not display its values in the report output.

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:

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 report 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:

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

In this section:

Controlling Formatting of Reports 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 [Selecting Records for Your Report](#) on page 157.

- 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 FIRST_NAME
BY DEPARTMENT
BY LAST_NAME
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 BY DEPARTMENT as the first sort condition, and other sort conditions may be added as needed.

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:

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

Controlling Formatting of Reports With Multiple Display Commands

How to:

Control the Format of Reports With Multiple Display Commands

Style a Report With SET DUPLICATECOL=ON

You can use the SET DUPLICATECOL command to reformat report requests that use multiple display commands, placing aggregated fields in the same column above the displayed field.

By default, each new display command in a request generates additional sort field and display field columns. With DUPLICATECOL set to OFF, each field occupies only one column in the request, with the values from each display command stacked under the values for the previous display command.

Syntax: How to Control the Format of Reports With Multiple Display Commands

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

where:

ON

Displays the report with each field as a column. This is the default value.

OFF

Displays the report with common fields as a row.

Example: Displaying Reports With Multiple Display Commands

The following request sums current salaries and education hours for the entire EMPLOYEE data source and for each department:

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

With DUPLICATECOL=ON, the output has separate columns for the grand totals and for the departmental totals:

CURR_SAL	ED_HRS	DEPARTMENT	CURR_SAL	ED_HRS
-----	-----	-----	-----	-----
\$222,284.00	351.00	MIS	\$108,002.00	231.00
		PRODUCTION	\$114,282.00	120.00

With DUPLICATECOL=OFF, the output has one column for each field. The grand totals are on the top row of the report, and the departmental totals are on additional rows below the grand totals:

DEPARTMENT	CURR_SAL	ED_HRS
-----	-----	-----
	\$222,284.00	351.00
MIS	\$108,002.00	231.00
PRODUCTION	\$114,282.00	120.00

The following request adds a PRINT command sorted by department and by last name to the previous request:

```
SET SPACES = 1
TABLE FILE EMPLOYEE
SUM CURR_SAL ED_HRS
SUM CURR_SAL ED_HRS BY DEPARTMENT AS 'DEPT'
PRINT FIRST_NAME CURR_SAL ED_HRS BY DEPARTMENT BY LAST_NAME
END
```

With DUPLICATECOL=ON, the output has separate columns for the grand totals, for the departmental totals, and for each last name:

CURR_SAL	ED_HRS	DEPT	CURR_SAL	ED_HRS	LAST_NAME	FIRST_NAME	CURR_SAL	ED_HRS
-----	-----	-----	-----	-----	-----	-----	-----	-----
\$222,284.00	351.00	MIS	\$108,002.00	231.00	BLACKWOOD	ROSEMARIE	\$21,780.00	75.00
					CROSS	BARBARA	\$27,062.00	45.00
					GREENSPAN	MARY	\$9,000.00	25.00
					JONES	DIANE	\$18,480.00	50.00
					MCCOY	JOHN	\$18,480.00	.00
					SMITH	MARY	\$13,200.00	36.00
		PRODUCTION	\$114,282.00	120.00	BANNING	JOHN	\$29,700.00	.00
					IRVING	JOAN	\$26,862.00	30.00
					MCKNIGHT	ROGER	\$16,100.00	50.00
					ROMANS	ANTHONY	\$21,120.00	5.00
					SMITH	RICHARD	\$9,500.00	10.00
					STEVENS	ALFRED	\$11,000.00	25.00

With `DUPLICATECOL=OFF`, the output has one column for each field. The grand totals are on the top row of the report, the departmental totals are on additional rows below the grand totals, and the values for each last name are on additional rows below their departmental totals:

DEPT	LAST_NAME	FIRST_NAME	CURR_SAL	ED_HRS
----	-----	-----	-----	-----
			\$222,284.00	351.00
MIS			\$108,002.00	231.00
	BLACKWOOD	ROSEMARIE	\$21,780.00	75.00
	CROSS	BARBARA	\$27,062.00	45.00
	GREENSPAN	MARY	\$9,000.00	25.00
	JONES	DIANE	\$18,480.00	50.00
	MCCOY	JOHN	\$18,480.00	.00
	SMITH	MARY	\$13,200.00	36.00
PRODUCTION			\$114,282.00	120.00
	BANNING	JOHN	\$29,700.00	.00
	IRVING	JOAN	\$26,862.00	30.00
	MCKNIGHT	ROGER	\$16,100.00	50.00
	ROMANS	ANTHONY	\$21,120.00	5.00
	SMITH	RICHARD	\$9,500.00	10.00
	STEVENS	ALFRED	\$11,000.00	25.00

Syntax: **How to Style a Report With SET DUPLICATECOL=ON**

In a StyleSheet, you can identify the rows you want to style by specifying which display command created those rows:

`VERBSET = n`

where:

`n`

Is the ordinal number of the display command in the report request.

Example: **Styling Rows Associated With a Specific Display Command**

The following request has two display commands:

1. `SUM CURR_SAL ED_HRS BY DEPARTMENT (totals by department).`

2. PRINT FIRST_NAME CURR_SAL ED_HRS BY DEPARTMENT BY LAST_NAME (values by employee by department).

```
SET DUPLICATECOL = OFF
TABLE FILE EMPLOYEE
SUM CURR_SAL ED_HRS BY DEPARTMENT
PRINT FIRST_NAME CURR_SAL ED_HRS BY DEPARTMENT BY LAST_NAME          ON TABLE
  HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE = REPORT, COLUMN= P4, VERBSET = 1, STYLE = ITALIC,      COLOR=BLUE,$
TYPE = REPORT, COLUMN= B2, VERBSET = 2, STYLE = UNDERLINE, COLOR = RED,$
ENDSTYLE
END
```

On the output:

- ❑ The fourth displayed column (P4, department total of CURR_SAL) for the SUM command is italic and blue.
- ❑ The second BY field (LAST_NAME) for the PRINT command is underlined and red.

When you style specific columns, using P notation means that you count every column that displays on the report output, including BY columns. Therefore, P1 is the DEPARTMENT column, P2 is the LAST_NAME column (this is also B2, the second BY field column), P3 is the FIRST_NAME column, P4 is the displayed version of the CURR_SAL column (the internal matrix has multiple CURR_SAL columns), and P5 is the displayed ED_HRS column (the internal matrix has multiple ED_HRS columns).

The output is:

PAGE 1				
DEPARTMENT	LAST_NAME	FIRST_NAME	CURR_SAL	ED_HRS
MIS			<i>\$108,002.00</i>	231.00
	<u>BLACKWOOD</u>	ROSEMARIE	\$21,780.00	75.00
	<u>CROSS</u>	BARBARA	\$27,062.00	45.00
	<u>GREENSPAN</u>	MARY	\$9,000.00	25.00
	<u>JONES</u>	DIANE	\$18,480.00	50.00
	<u>MCCOY</u>	JOHN	\$18,480.00	.00
	<u>SMITH</u>	MARY	\$13,200.00	36.00
PRODUCTION			<i>\$114,282.00</i>	120.00
	<u>BANNING</u>	JOHN	\$29,700.00	.00
	<u>IRVING</u>	JOAN	\$26,862.00	30.00
	<u>MCKNIGHT</u>	ROGER	\$16,100.00	50.00
	<u>ROMANS</u>	ANTHONY	\$21,120.00	5.00
	<u>SMITH</u>	RICHARD	\$9,500.00	10.00
	<u>STEVENS</u>	ALFRED	\$11,000.00	25.00

Improving Efficiency With External Sorts

In this section:

Providing an Estimate of Input Records or Report Size for Sorting

Mainframe External Sort Utilities and Message Options

Aggregation by External Sort (Mainframe Environments Only)

Changing Retrieval Order With Aggregation

Creating a HOLD File With an External Sort

How to:

Determine the Type of Sort Used

Control External Sorting

Query the Sort Type

Reference:

Requirements for External Sorting

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 180 to 200K, although many factors affect the size of the data that can be handled by the internal sort.

The FOCSORT file used for the internal sort can grow to any size allowed by the operating system running and the available disk space. The user does not have to break a request up to accommodate massive files. In previous releases, the FOCSORT file was limited to 2 GB and the user received a FOC298 message when the FOCUS limit was exceeded. With no limit enforced by FOCUS, the operating system provides whatever warning and error handling it has for the management of a FOCSORT file that exceeds its limits.

You can generate larger reports somewhat faster by using dedicated sorting products, such as SyncSort, or DFSORT.

To use an external sort, the EXTSORT parameter must be ON. Use of a StyleSheet turns off external sorting.

Note that in Mainframe environments, external sorting is supported with the French, Spanish, German, and Scandinavian National Languages (Swedish, Danish, Finnish, and Norwegian). 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 DFSORT and SyncSort external sort products with any TABLE, FML, GRAPH, or MATCH request.

Procedure: How to Determine the Type of Sort Used

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, the external sort utility will be invoked.

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 supplied data already in order.

EXTERNAL

An external sorting product sorted the report.

NONE

The report did not require sorting.

Providing an Estimate of Input Records or Report Size for Sorting

How to:

Provide an Estimate of Input Records or Report Size for Sorting

There are two advantages to providing an estimate for the input size (ESTRECORDS) or the report size (ESTLINES):

- ❑ If the request cannot be converted to a TABLEF request and the file size estimate shows that the external sort will be needed, FOCUS initiates the external sort immediately, which makes a FOCUS merge unnecessary. Without the estimate, such a request always performs this merge.
- ❑ In Mainframe environments, FOCUS passes the file size to the external sort, which enables it to allocate work files of the appropriate size.

Syntax: **How to Provide an Estimate of Input Records or Report Size for Sorting**

From the command line:

```
SET ESTRECORDS = n  
SET ESTLINES = n
```

In a request:

```
ON TABLE SET ESTRECORDS n ON TABLE SET ESTLINES n
```

where:

n

Is the estimated number of records or lines to be sorted.

Note: These parameters cannot be set in FOCPARM.

Mainframe External Sort Utilities and Message Options

In this section:

Diagnosing External Sort Errors

How to:

Select a Sort Utility and Message Options

By default, error messages created by a Mainframe external sort product are not displayed. However, you may wish to display these messages on your screen for diagnostic purposes.

Procedure: How to Select a Sort Utility and Message Options

You use the SET SORTLIB command to both specify the sort utility used at your site and, for DFSORT and SYNCSORT on z/OS, to display sort messages.

1. Issue the SET SORTLIB command to specify the sort utility being used:

```
SET SORTLIB = {sortutility|DEFAULT}
```

where:

sortutility

Can be one of the following:

VMSORT for VMSORT on z/VM. On z/OS, this becomes DFSORT.

DFSORT for DFSORT on z/VM or DFSORT without messages on z/OS.

MVSMSGDF for DFSORT on z/VM or DFSORT with messages on z/OS.

SYNCSORT for SyncSort on z/VM or SyncSort without messages on z/OS.

MVSMSGSS for SyncSort on z/VM or SyncSort with standard messages on z/OS.

MVSMSGSD for SyncSort on z/VM or SyncSort with debug (verbose) messages on z/OS.

DEFAULT for VMSORT on z/VM or DFSORT on z/OS. However, It is more efficient and highly recommended that you explicitly specify the sort utility using one of the other values.

2. If you specified a sort option that produces sort messages on z/OS, you must direct the sort messages to the batch output stream, a file, or the terminal. On z/VM, you must FILEDEF DDNAMEs SYSOUT and SORTTRACE in order to generate sort messages.

Allocate DDNAME SYSOUT to the batch output stream or a file on z/OS by inserting the appropriate following DD card into your FOCUS batch JCL, if it is not already there. For example, the following DD card allocates DDNAME SYSOUT to the batch output stream:

```
//SYSOUT DD SYSOUT=*
```

Online, the following ALLOCATE command allocates DDNAME SYSOUT to the terminal:

```
ALLOC F(SYSOUT) DA(*) REU
```

On z/VM, you must FILEDEF DDNAMEs SORTTRACE (to generate sort messages) and DDNAME SYSOUT (to print the messages) after issuing the SET SORTLIB command to specify the sort utility.

If you want to send sort messages to the terminal on z/VM, Issue the following FILEDEF commands after issuing the SET SORTLIB command to specify the sort utility:

```
CMS FILEDEF SYSOUT TERM ( PERM  
CMS FILEDEF SORTTRACE TERM ( PERM
```

Diagnosing External Sort Errors

When an external sort generates an error, you can generate a trace of sort processing and examine the FOCUS return codes and messages to diagnose the problem.

Procedure: How to Trace Sort Processing

When an external sort problem occurs, one of the following messages is generated:

```
(FOC909) CRITICAL ERROR IN EXTERNAL SORT. RETURN CODE IS: xxxx  
(FOC1810) External sort not found  
(FOC1899) Load of %1 (external-sort module) under %2 failed
```

In response to these messages, as well as for any other problem with sorting, it is useful to trace sort processing. For information on diagnosing external sort problems, see [Diagnosing External Sort Errors](#) on page 150.

1. Allocate (on z/OS) or FILEDEF (on z/VM) DDNAME FSTRACE to the terminal or a file. The following example sends trace output to the terminal:

```
//FSTRACE DD SYSOUT=*,DCB=(RECFM=FA,LRECL=133,BLKSIZE=133)
```

or

```
FILEDEF FSTRACE TERM (RECFM FA LRECL 133 BLKSIZE 133 PERM
```

2. Activate the trace by adding the following commands in any supported profile or a FOCEXEC:

```
SET TRACEUSER = ON
SET TRACEON   = SORT/1/FSTRACE
```

Reference: External Sort Messages and Return Codes

When you receive a FOC909 message, it includes a return code:

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

You may also receive one of the following messages:

```
(FOC1810) External sort not found
(FOC1899) Load of %1 (external-sort module) under %2 failed
```

The following notes apply when this message or a FOC1800 or FOC1899 message is generated by a TABLE request:

- ❑ The most common value for xxxx is 16. However, return code 16 is issued for a number of problems, including but not limited to the following:
 - ❑ Syntax errors.
 - ❑ Memory shortage.
 - ❑ I/O errors (depending on installation options).
 - ❑ Space problems with output.
 - ❑ Space problems with work files.

In order to diagnose the error, you must generate external sort messages (using the instructions in [How to Select a Sort Utility and Message Options](#) on page 149 and [How to Trace Sort Processing](#) on page 150) and then reproduce the failure.

For return codes not described below, follow the same procedure described for return code 16.

- ❑ Return code 20 is issued by DFSORT under z/OS if messages were requested (using the MVSMSGDJ option of the SET SORTLIB command), but the SYSOUT DD card is missing. DFSORT terminates after issuing the return code. Under the same conditions, SyncSort attempts to open SYSOUT, producing the following message, and then continues with messages written to the operator or terminal:

```
IEC130I SYSOUT DD STATEMENT MISSING.
```

- ❑ On z/VM, return code 24 or a FOC1810 message means that the external sort could not be found. Take the following steps:

1. Global the sort library before executing FOCUS.

One of the following examples may apply:

```
GLOBAL TXTLIB VMSLIB
```

or

```
GLOBAL TXTLIB SYNCSORT
```

2. Ensure that the FOCADLIB EXEC is accessed. This EXEC is usually on the FOCUS production disk. This is the EXEC used by the sort routines to GLOBAL the SORT TXTLIBS.
3. The sort library SORTLIB, if found, will automatically be accessed. If no SORTLIB TXTLIB is accessed before executing FOCUS, this might be the problem. Make a copy of the current sort TXTLIB named as SORTLIB TXTLIB on any accessed disk.

- ❑ Return code 36 or a FOC1899 message under z/OS means that the external sort module could not be found; check the STEPLIBs allocated.

When REBUILD INDEX invokes an external sort that fails, it generates a message similar to the following:

```
ERROR OCCURRED IN THE SORT yyyyyyyzzzzzzzz
```

In this case, the return code is yyyyyyy and it is expressed in hex. The final eight digits (zzzzzzzz) should be ignored.

Translate the return code into decimal and follow the instructions for return codes in a TABLE request.

Note also that when a TABLE request generates a non-zero return code from an external sort, FOCUS is terminated. By contrast, when REBUILD INDEX gets a non-zero return code from an external sort, the REBUILD command is terminated but FOCUS continues.

Reference: Responding to an Indication of Inadequate Sort Work Space

Before following these instructions, make sure that external sort messages were generated (for information, see [How to Select a Sort Utility and Message Options](#) on page 149) and that they clearly show that the reason for failure was inadequate sort work space.

1. Make an estimate of the number of lines of output the request will produce.
2. Set the ESTLINES parameter in the request or FOCEXEC. For information, see [Providing an Estimate of Input Records or Report Size for Sorting](#) on page 148.

FOCUS will pass this estimate to the external sort utility through the parameter list.

Do not override the DD cards for SORTWKnn, S001WKnn, DFSPARM, or \$ORTPARM without direct instructions from technical support. The instructions in [How to Select a Sort Utility and Message Options](#) on page 149, [How to Trace Sort Processing](#) on page 150, and [Providing an Estimate of Input Records or Report Size for Sorting](#) on page 148 should provide equivalent capabilities.

Aggregation by External Sort (Mainframe Environments Only)

How to:

Use Aggregation in Your External Sort

Reference:

Usage Notes for Aggregating With an 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. (If you must test &LINES, do not use this feature.)

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 use SyncSort or DFSORT.
- ❑ Your query should be simple (that is, it should be able to take advantage of the TABLEF facility).

- ❑ 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 an External Sort 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. However, you can control the order of display using the SUMPREFIX parameter. With SUMPREFIX = LST (the default), the last instance displays even with EXTAGGR = ON.

The following command turns aggregation ON and leaves SUMPREFIX set to LST (the default) and, therefore, displays the last record:

```
SET EXTAGGR = ON
SET SUMPREFIX = LST
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: SUMPREFIX is described in [Changing Retrieval Order With Aggregation](#) on page 155.

With SUMPREFIX = FST, the output is:

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

Changing Retrieval Order With Aggregation

How to:

Set Retrieval Order

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.

Creating a HOLD File With an External Sort

How to:

Create HOLD Files With an External Sort

Reference:

Usage Notes for Creating a HOLD File With an External Sort

You can use Mainframe external sort packages to create HOLD files, producing substantial savings 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 a HOLD File With an External Sort**

- ❑ The default setting of EXTSORT=ON must be in effect.
- ❑ EXTHOLD must be ON.
- ❑ The request must contain a BY field.
- ❑ The type of HOLD file created must be a FOCUS, XFOCUS, ALPHA, or BINARY file.
- ❑ Your query should be simple. AUTOTABLEF analyzes a query and determines whether the combination of display commands and formatting options requires the internal matrix. In cases where it is determined that a matrix is not necessary to satisfy the query, you may avoid the extra internal costs associated with creating the matrix. The internal matrix is stored in a file or data set named FOCSORT. The AUTOTABLEF default is ON, in order to realize performance gains.
- ❑ 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.

5 | Selecting Records for Your Report

When generating a report and selecting fields, you may not want to include 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 data that you can easily redefine each time you issue the report request.

Topics:

- ❑ Selecting Records Overview
- ❑ 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

Selecting Records Overview

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 159.
- ❑ The aggregate value of a field (for example, the sum or average of field values). See [Selection Based on Aggregate Values](#) on page 167.
- ❑ The existence of missing values for a field, whether field values fall within a range, or whether a field does not contain a certain value. See [Types of Record Selection Tests](#) on page 174.
- ❑ The number of records that exist for a field (for example, the first 50 records), rather than on the field values. See [Setting Limits on the Number of Records Read](#) on page 189.
- ❑ 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 188.

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. It is recommended 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 190.

Selections Based on Individual Values

In this section:

Controlling Record Selection in Multi-path Data Sources

How to:

Select Records With WHERE

Reference:

Usage Notes for WHERE Phrases

The WHERE phrase selects records from the data source 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 161. For additional information, see [Using Compound Expressions for Record Selection](#) on page 169.

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 162.

Syntax: **How to Select Records With WHERE**

```
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 [Using Expressions](#) on page 323. 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 171.

;

Is an optional semicolon that 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. If a field name is enclosed in single or double quotation marks, it is treated as a literal string, not a field reference.
- ❑ The operators EQ, NE, GE, GT, LT, LE, CONTAINS, OMMITS, FROM ... TO, NOT-FROM ... TO, INCLUDES, EXCLUDES, LIKE, and NOT LIKE.
- ❑ All arithmetic operators (+, -, *, /, **), as well as, functions (MIN, MAX, ABS, and SQRT).
- ❑ 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*).
- ❑ A date literal used in a selection test against a date field cannot contain the day of the week value.
- ❑ Text fields. However, the only operators supported for use with text fields are CONTAINS and OMMITS.
- ❑ All functions.

You can build complex selection criteria by joining simple expressions with AND and OR logical operators and, optionally, adding parentheses to specify explicitly 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 169.

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 are 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; all other records are ignored.

The output is:

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. This request uses multiple WHERE phrases so that only those employees in the MIS or Production departments with the last name of Cross or Banning are included in the report.

```
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:

EMP_ID	LAST_NAME
119329144	BANNING
818692173	CROSS

For related information, see *Using Compound Expressions for Record Selection* on page 169.

Controlling Record Selection in Multi-path Data Sources

How to:

Control Record Selection in Multi-path Data Sources

Reference:

Requirements and Usage Notes for MULTIPATH = COMPOUND

MULTIPATH and SET ALL Combinations

Rules for Determining If a Segment Is Required

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.
- ❑ 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, and 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, use

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

To set MULTIPATH in a report request, use

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

where:

[SIMPLE](#)

Is the default value. 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.

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

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

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.

For related information, see [MULTIPATH and SET ALL Combinations](#) on page 165 and [Rules for Determining If a Segment Is Required](#) on page 166.

Reference: Requirements and Usage Notes for MULTIPATH = COMPOUND

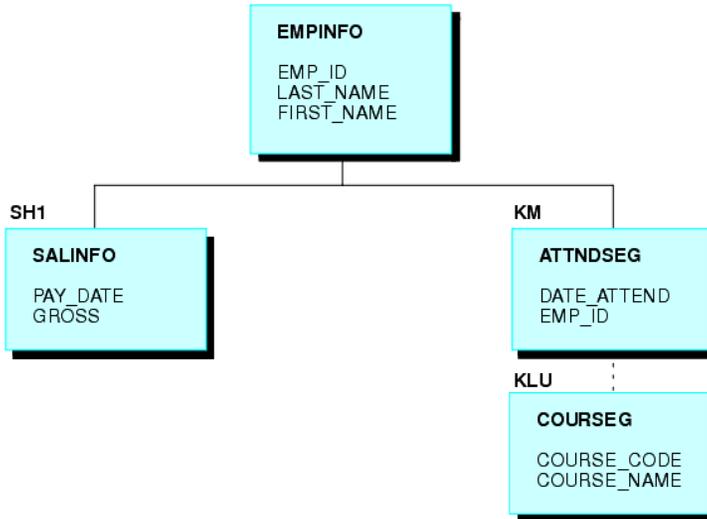
- ❑ The minimum memory requirement for the MULTIPATH = COMPOUND setting is 4K per active segment. If there is insufficient memory, the SIMPLE setting is implemented and a message is returned.

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.

The output is:

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 FOR PROG
MCKNIGHT	ROGER	\$1,342.00	.	.
ROMANS	ANTHONY	\$1,760.00	.	.
SMITH	MARY	\$1,100.00	81/11/16	BASIC REPORT PREP FOR PROG
	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.

The output is:

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

Reference: 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	MULTIPATH=COMPOUND
<pre>SET ALL = OFF PRINT EMP_ID PAY_DATE DATE_ATTEND</pre>	Shows employees who have either SALINFO data or ATTNDSEG data.	Shows employees who have both SALINFO and ATTNDSEG data.
<pre>SET ALL = ON PRINT EMP_ID PAY_DATE DATE_ATTEND</pre>	Shows employees who have SALINFO data or ATTNDSEG data or no child data at all.	Same as SIMPLE.
<pre>SET ALL = OFF PRINT EMP_ID PAY_DATE DATE_ATTEND WHERE PAY_DATE EQ 980115</pre>	Shows employees who have either SALINFO data for 980115 or any ATTNDSEG data. Produces (FOC144) message.	Shows employees who have both SALINFO data for 980115 and ATTNDSEG data.

Request	MULTIPATH=SIMPLE	MULTIPATH=COMPOUND
<pre>SET ALL = ON PRINT EMP_ID PAY_DATE DATE_ATTEND WHERE PAY_DATE EQ 980115</pre>	<p>Shows employees who have <i>either</i> SALINFO data for 980115 <i>or</i> any ATTNDSEG data.</p> <p>Produces (FOC144) message.</p>	<p>Shows employees who have SALINFO data for 980115. Any DATE_ATTEND data is also shown.</p>
<pre>SET ALL = OFF PRINT ALL.EMP_ID DATE_ATTEND WHERE PAY_DATE EQ 980115</pre>	<p>Shows employees who have <i>either</i> SALINFO data for 980115 <i>or</i> any ATTNDSEG data.</p> <p>Produces (FOC144) message.</p>	<p>Shows employees who have SALINFO data for 980115. Any DATE_ATTEND data is also shown.</p>
<pre>SET ALL = ON or OFF PRINT EMP_ID PAY_DATE DATE_ATTEND WHERE PAY_DATE EQ 980115 AND COURSE_CODE EQ '103'</pre>	<p>Shows employees who have <i>either</i> SALINFO data for 980115 <i>or</i> COURSE 103.</p> <p>Note: SIMPLE treats AND in the WHERE clause as OR.</p> <p>Produces (FOC144) message.</p>	<p>Shows employees who have <i>both</i> SALINFO data for 980115 <i>and</i> COURSE 103.</p>

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

For related information about the ALL parameter, see [Handling Records With Missing Field Values](#) on page 807.

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 existence depends on the child segment existence, and the child segment depends on the grandchild 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 [Handling Records With Missing Field Values](#) on page 807, and the *Describing Data* manual.

Selection Based on Aggregate Values

How to:

Select Records With WHERE TOTAL

Reference:

Usage Notes for WHERE TOTAL

You can select records based on the aggregate value of a field. For example, on the sum of field values, or on the average of field values, by using the WHERE TOTAL phrase. WHERE TOTAL is very helpful when you employ the aggregate display commands SUM and COUNT, and is required for fields with a 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 example, see [Using WHERE TOTAL for Record Selection](#) on page 168.

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 [Using Expressions](#) on page 323. 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 171.

;

Is an optional semicolon that 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.

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:

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:

DEPARTMENT	CURR_SAL
PRODUCTION	\$114,282.00

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:

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.

You should always use parentheses in complex expressions to ensure that the expression is evaluated correctly. For example:

```
WHERE (SEATS EQ 2) AND (SEATS NOT-FROM 3 TO 4)
```

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 Record Selection 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 are excluded, the logical AND is 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:

LAST_NAME	CURR_SAL
CROSS	\$27,062.00

Using Operators in Record Selection Tests

Reference:

Operators Supported for WHERE and IF 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.

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.

WHERE Operator	IF Operator	Meaning
FROM <i>lower</i> TO upper		Tests for and selects values within a range of values.
IS-FROM lower TO upper	IS-FROM lower TO upper	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.
NOT-FROM lower TO upper	NOT-FROM lower TO upper	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. If some instances of the field contain no data, they have missing data. For information on missing data, see Handling Records With Missing Field Values on page 807.
CONTAINS LIKE	CONTAINS LIKE	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 UNLIKE	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.

WHERE Operator	IF Operator	Meaning
EXCLUDES	EXCLUDES	Tests whether a chain of values of a given field in a child segment excludes all of a list of literals.
IN (z,x,y)		Selects records based on values found in an unordered list.
NOT ... IN (z,x,y)		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

In this section:

Range Tests With FROM and TO

Range Tests With GE and LE or GT and LT

Missing Data Tests

Character String Screening With CONTAINS and OMITS

Screening on Masked Fields With LIKE and IS

Using an Escape Character for LIKE

Qualifying Parent Segments Using INCLUDES and EXCLUDES

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 171. You can test for:

- ❑ Values that lie within or outside of a range. See [Range Tests With FROM and TO](#) on page 175 and [Range Tests With GE and LE or GT and LT](#) on page 176.
- ❑ Missing or existing data. See [Missing Data Tests](#) on page 178.
- ❑ The existence or absence of a character string. See [Character String Screening With CONTAINS and OMITS](#) on page 179.
- ❑ Partially defined character strings in a data field. See [Screening on Masked Fields With LIKE and IS](#) on page 180.

- ❑ Literals in a parent segment. See [Qualifying Parent Segments Using INCLUDES and EXCLUDES](#) on page 187.

Range Tests With FROM and TO

How to:

Specify a Range Test (FROM and TO)

Use the operators FROM ... TO and NOT-FROM ... TO in order to determine whether field values fall within or outside of 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)

```
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

The following illustrates how you can use the range test NOT-FROM ... TO to display only those records that fall outside of the specified range. In this example, it is all employees whose salaries do not fall in the range 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:

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

How to:
Specify Range Tests (GE and LE)

The operators GE (greater than or equal to), LE (less than or equal to), GT (greater than), and LT (less than) 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

```
WHERE fieldname GE lower AND fieldname LE upper
```

To find records whose values do not fall in a specified range, use

```
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

The following illustrates how you can select values that are outside a range of values using the LT and GT operators. In this example, only those employees whose salaries are less than \$12,000 and greater than \$22,000 are included in the output.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY LAST_NAME
WHERE CURR_SAL LT 12000 OR CURR_SAL GT 22000
END
```

The output is:

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

How to:

Test for Missing Data

Test for Existing Data

When creating report requests, you may want to test for missing data. This type of test is most useful when fields that have missing data also have the MISSING attribute set to ON in the Master File. For information on missing data, see [Handling Records With Missing Field Values](#) on page 807, and the *Describing Data* manual.

Syntax: How to Test for Missing Data

```
{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

```
{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 you do not know the exact spelling of a value. 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

The following examples illustrate several ways to use the CONTAINS and OMITS operators. The field name that is being tested must appear on the left side of the CONTAINS or OMITS operator.

- ❑ In this example, the characters JOHN are contained in JOHNSON, and are selected by the following phrase:

```
WHERE LAST_NAME CONTAINS 'JOHN'
```

The LAST_NAME field may contain the characters JOHN anywhere in the field.

- ❑ In this example, any last name without the string JOHN is selected:

```
WHERE LAST_NAME OMITS 'JOHN'
```

- ❑ In this example, all names that contain the letters ING are retrieved.

```
TABLE FILE EMPLOYEE
LIST LAST_NAME AND FIRST_NAME
WHERE LAST_NAME CONTAINS 'ING'
END
```

The output is:

LIST	LAST_NAME	FIRST_NAME
1	BANNING	JOHN
2	IRVING	JOAN

Screening on Masked Fields With LIKE and IS

How to:

Screen Fields Based on a Mask (Using LIKE and NOT LIKE)

Screen Using LIKE and UNLIKE in an IF Phrase

Screen Fields Based on a Mask (Using IS and IS-NOT)

Reference:

Restrictions on Masking Characters

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 wildcard characters for screening on masked fields with:

- ❑ LIKE and NOT LIKE operators are % and _. The percent allows any following sequence of zero or more characters. The underscore indicates that any character in that position is acceptable. The LIKE operator is supported in expressions that are used to derive temporary fields with either the DEFINE or COMPUTE command.
- ❑ IS (or EQ) and IS-NOT (or NE) operators are \$ and \$*. The dollar sign indicates that any character in that position is acceptable. The \$* is shorthand for writing a sequence of dollar signs to fill the end of the mask without specifying a length. This combination can only be used at the end of the mask.

Note: The IS (or EQ) and IS-NOT (or NE) operators support screening based on a mask for fixed length formats only. If the format is a variable length format, for example, AnV, use the LIKE or NOT LIKE operator to screen based on a mask.

Syntax: **How to Screen Fields Based on a Mask (Using LIKE and NOT LIKE)**

To search for records with the LIKE operator, use

```
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

Is 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, and the percent sign (%) allows any following sequence of zero or more characters.

For related information, see [Restrictions on Masking Characters](#) on page 182.

Syntax: How to Screen Using LIKE and UNLIKE in an IF Phrase

To search for records with the LIKE operator, use

```
IF field LIKE 'mask1' [OR 'mask2' ...]
```

To reject records based on the mask value, use

```
IF field UNLIKE 'mask1' [OR 'mask2' ...]
```

where:

field

Is any valid field name or alias.

mask1, mask2

Are the alphanumeric patterns you want to use for comparison. The single quotation marks are required if the mask contains blanks. There are two wildcard characters that you can use in a mask. The underscore (_) indicates that any character in that position is acceptable, and the percent sign (%) allows any following sequence of zero or more characters. Every other character in the mask accepts only itself in that position as a match to the pattern.

Syntax: How to Screen Fields Based on a Mask (Using IS and IS-NOT)

To search for records with the IS operator, use

```
{WHERE|IF} field {IS|EQ} 'mask'
```

To reject records based on the mask value, use

```
{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

Is 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. The \$* is shorthand for writing a sequence of dollar signs to fill the end of the mask without specifying a specific length. This combination can only be used at the end of the mask.

For related information, see [Restrictions on Masking Characters](#) on page 182.

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:

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:

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:

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 following example illustrates how to screen on initial characters and specify the length of the field value you are searching for. In this example, the WHERE phrase states that the last name must begin with BAN and be 7 characters in length (the three initial characters BAN and the 4 placeholders, in this case the dollar sign). The remaining characters in the field (positions 8 through 15) must be blank.

```
TABLE FILE EMPLOYEE
PRINT  LAST_NAME
WHERE  LAST_NAME IS 'BAN$$$$'
END
```

The output is:

```
LAST_NAME  
-----  
BANNING
```

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 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

How to:

- Use an Escape Character in a WHERE Phrase
- Specify an Escape Character for a Mask in an IF Phrase

Reference:

- Usage Notes for Escape Characters

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 180.

Syntax: **How to Use an Escape Character in a WHERE Phrase**

Any single character can be used as an escape character, if prefaced with the word ESCAPE

```
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.

Syntax: How to Specify an Escape Character for a Mask in an IF Phrase

You can assign any single character as an escape character by prefacing it with the word ESCAPE in the LIKE or UNLIKE syntax

```
IF field {LIKE|UNLIKE} 'mask1' ESCAPE 'a' [OR 'mask2' ESCAPE 'b' ...
```

where:

field

Is any valid field name or alias to be evaluated in the selection test.

mask1, mask2

Are search patterns that you supply. The single quotation marks are required.

a, b ...

Are single characters that you identify as escape characters. Each mask can specify its own escape character or use the same character as other masks. 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 if the mask contains blanks.

Reference: Usage Notes for Escape Characters

- ❑ The use of an escape character in front of any character other than %, _, and itself is ignored.
- ❑ The escape character itself can be escaped, thus becoming a normal character in a string (for example, 'abc%\%\\').
- ❑ Only one escape character can be used per LIKE phrase in a WHERE phrase.
- ❑ 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.
- ❑ If a WHERE criterion is used with literal OR phrases, the ESCAPE must be on the first OR phrase, and applies 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 in a WHERE Phrase

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

Example: Using an Escape Character in an IF Phrase

The VIDEOTR2 data source contains an e-mail address field. To search for e-mail addresses with the characters 'handy_' you can issue the following request:

```
TABLE FILE VIDEOTR2
PRINT CUSTID LASTNAME FIRSTNAME EMAI
IF 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 EMAI
IF 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

Reference:

Usage Notes for INCLUDES and EXCLUDES

You can test whether instances of a given field in a child segment include or exclude all literals in a list using the INCLUDES and EXCLUDES operators. 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.

Note: INCLUDES and EXCLUDES work only with multi-segment FOCUS data sources.

Reference: Usage Notes for INCLUDES and EXCLUDES

- ❑ Literals containing embedded blanks must be enclosed in single quotation marks.
- ❑ The total number of literals must be 31 or less.
- ❑ 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, the ORs are changed to ANDs.

Example: Selecting Records With INCLUDES and EXCLUDES

A request that contains the phrase

```
WHERE JOBCODE INCLUDES A01 OR B01
```

returns 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 is not 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:

```
COUNTRY
-----
ENGLAND
```

Selections Based on Group Key Values

Some 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 group key 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 169.

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'
```

Setting Limits on the Number of Records Read

How to:

Limit 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 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 would 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

```
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

```
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:

LAST_NAME	FIRST_NAME	EMP_ID
STEVENS	ALFRED	071382660
SMITH	MARY	112847612
JONES	DIANE	117593129
SMITH	RICHARD	119265415

Selecting Records Using IF Phrases

How to:
Select Records Using the IF Phrase

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 171.

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

```
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 171.

literal

Can be the MISSING keyword (as described in [Missing Data Tests](#) on page 178) 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 [Using Expressions](#) on page 323. You can accomplish this more easily with WHERE. See [Using Compound Expressions for Record Selection](#) on page 169.

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:

```
EMP_ID      LAST_NAME
-----
818692173  CROSS
```

Reading Selection Values From a File

How to:

- Read Selection Values From a File (WHERE)
- Read Selection Values From a File (IF)

Reference:

- Usage Notes for Reading 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. For information on HOLD and SAVE files, see [Saving and Reusing Your Report Output](#) on page 421.

Note that in z/OS, 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)**

```
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.

file

Is the name of the file.

For z/OS, 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 193.

Syntax: **How to Read Selection Values From a File (IF)**

```
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 171).

file

Is the name of the file.

For z/OS, this is the ddname assigned by a DYNAM or TSO ALLOCATE command.

For CMS, this is the ddname assigned by a FILEDEF command.

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. Actual literals may also be mixed with the file names. For example:

IF filename 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 are 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 specifies 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)

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 are selected:

```
TABLE FILE GGPRODS  
SUM UNIT_PRICE  
BY PRODUCT_DESCRIPTION  
IF PRODUCT_ID IS (EXPER)  
END
```

If you include the selection criteria directly in the request, the IF phrase specifies the values explicitly:

```
IF PRODUCT_DESCRIPTION EQ 'B141' or 'B142'
```

The output is:

Product	Unit Price
-----	-----
French Roast	81.00
Hazelnut	58.00

Assigning Screening Conditions to a File

In this section:

Preserving Filters Across Joins

How to:

Declare a Filter

Activate or Deactivate Filters

Query the Status of Filters

Reference:

Usage Notes for Virtual Fields Used in Filters

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 196.

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 198.

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.

CLEAR

Deletes any existing filter phrases, including any previously defined virtual fields.

ADD

Enables you to add new filter phrases to an existing filter declaration without clearing previously defined filters.

filter-defines

Are virtual fields declared for use in filters. For more information, see [Usage Notes for Virtual Fields Used in Filters](#) on page 196.

filtername1...filternamen

Is the name by which the filter is referenced in subsequent SET FILTER commands. This name may be up to eight characters long 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, or 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.
- ❑ Cannot reuse a virtual field name for a the same file.

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= {*_|xx|yy zz} IN file {ON|OFF}
```

where:

*

Denotes all declared filters. This is the default value.

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 to which you are assigning screening conditions.

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 that the filters contain, not to exceed the limit of WHERE/IF criteria in any single report request.

OFF

Deactivates all (*) or specifically named filters for the data source. OFF is the default value.

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

To determine the status of existing filters, use

```
? 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 example queries 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.

Preserving Filters Across Joins

How to:

Preserve Filter Definitions With KEEPFILTERS

By default, filters defined on the host data source are cleared by a JOIN command. However, filters can be maintained when a JOIN command is issued, by issuing the SET KEEPFILTERS=ON command.

Setting KEEPFILTERS to ON reinstates filter definitions and their individual declared status after a JOIN command. The set of filters and virtual fields defined prior to each join is called a context (see your documentation on SET KEEPDEFINES and on DEFINE FILE SAVE for information about contexts as they relate to virtual fields). Each new JOIN or DEFINE FILE command creates a new context.

If a new filter is defined after a JOIN command, it cannot have the same name as any previously defined filter unless you issue the FILTER FILE command with the CLEAR option. The CLEAR option clears all filter definitions for that data source in all contexts.

When a JOIN is cleared, each filter definition that was in effect prior to the JOIN command and that was not cleared, is reinstated with its original status. Clearing a join by issuing the JOIN CLEAR join_name command removes all of the contexts and filter definitions that were created after the JOIN join_name command was issued.

Syntax: How to Preserve Filter Definitions With KEEPFILTERS

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

where:

OFF

Does not preserve filters issued prior to a join. OFF is the default value.

ON

Preserves filters across joins.

Example: Preserving Filters With KEEPFILTERS

The first filter, UNITPR, is defined prior to issuing any joins, but after setting KEEPFILTERS to ON:

```
SET KEEPFILTERS = ON
FILTER FILE VIDEOTRK
PERUNIT/F5 = TRANSTOT/QUANTITY;
NAME=UNITPR
WHERE PERUNIT GT 2
WHERE LASTNAME LE 'CRUZ'
END
```

The ? FILTER command shows that the filter named UNITPR was created but not activated (activation is indicated by an asterisk in the SET column of the display:

```
? FILTER

SET FILE      FILTER NAME DESCRIPTION
-----
VIDEOTRK UNITPR
```

Next the filter is activated:

```
SET FILTER= UNITPR IN VIDEOTRK ON
```

The ? FILTER query shows that the filter is now activated:

```
? FILTER

SET FILE      FILTER NAME DESCRIPTION
-----
*  VIDEOTRK UNITPR
```

The following TABLE request is issued against the filtered data source:

```
TABLE FILE VIDEOTRK
SUM QUANTITY TRANSTOT BY LASTNAME
END
```

The output shows that the TABLE request retrieved only the data that satisfies the UNITPR filter:

```
NUMBER OF RECORDS IN TABLE=          6  LINES=          3
ACCESS LIMITED BY FILTERS

PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

LASTNAME      QUANTITY  TRANSTOT
-----
CHANG          3         31.00
COLE           2         18.98
CRUZ           2         16.00
```

Now, the VIDEOTRK data source is joined to the MOVIES data source. The ? FILTER query shows that the join did not clear the UNITPR filter:

```
JOIN MOVIECODE IN VIDEOTRK TO ALL MOVIECODE IN MOVIES AS J1
```

The ? FILTER command shows that the UNITPR filter still exists and is still activated:

```
? FILTER
```

```
SET FILE      FILTER NAME DESCRIPTION
-----
```

```
*  VIDEOTRK UNITPR
```

Next a new filter, YEARS1, is created and activated for the join between VIDEOTRK and MOVIES:

```
FILTER FILE VIDEOTRK
YEARS/I5 = (EXPDATE - TRANSDATE)/365;
NAME=YEARS1
WHERE YEARS GT 1
END
SET FILTER= YEARS1 IN VIDEOTRK ON
```

The ? FILTER query shows that both the UNITPR and YEARS1 filters exist and are activated:

```
? FILTER
```

```
SET FILE      FILTER NAME DESCRIPTION
-----
```

```
*  VIDEOTRK UNITPR
```

```
*  VIDEOTRK YEARS1
```

Now, J1 is cleared. The output of the ? FILTER command shows that the YEARS1 filter that was created after the JOIN command was issued no longer exists. The UNITPR filter created prior to the JOIN command still exists with its original status:

```
JOIN CLEAR J1
```

```
? FILTER
```

```
SET FILE      FILTER NAME DESCRIPTION
-----
```

```
*  VIDEOTRK UNITPR
```

VSAM Record Selection Efficiencies

In this section:

Reporting From Files With Alternate Indexes

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 reading 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, equality logic is used to 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
```

It then 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.

Note: It is not necessary to take an explicit indexed view to use the index.

6 | Creating Temporary Fields

When you create a report, you are not restricted to the fields that exist in your data source. If you can generate the information you want from the existing data, you can create a temporary field to evaluate and display it. A temporary field takes up no storage space in the data source. It is created only when needed.

Topics:

- ❑ What Is a Temporary Field?
- ❑ Defining a Virtual Field
- ❑ Creating a Calculated Value
- ❑ Assigning Column Reference Numbers
- ❑ Calculating Trends and Predicting Values With FORECAST
- ❑ Calculating Trends and Predicting Values With Multivariate REGRESS
- ❑ Using Text Fields in DEFINE and COMPUTE
- ❑ Creating Temporary Fields Independent of a Master File

What Is a Temporary Field?

Reference:

Types of Temporary Fields

Evaluation of Temporary Fields

Selecting a Temporary Field

A temporary field is a field whose value is not stored in the data source, but can be calculated from the data that is there, or assigned an absolute value. A temporary field takes up no storage space in the data source, and is created only when needed.

When you create a temporary field, you determine its value by writing an expression. You can combine fields, constants, and operators in an expression to produce a single value. For example, if your data contains salary and deduction amounts, you can calculate the ratio of deductions to salaries using the following expression:

```
deduction / salary
```

You can specify the expression yourself, or you can use one of the many supplied functions that 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.

Reference: **Types of Temporary Fields**

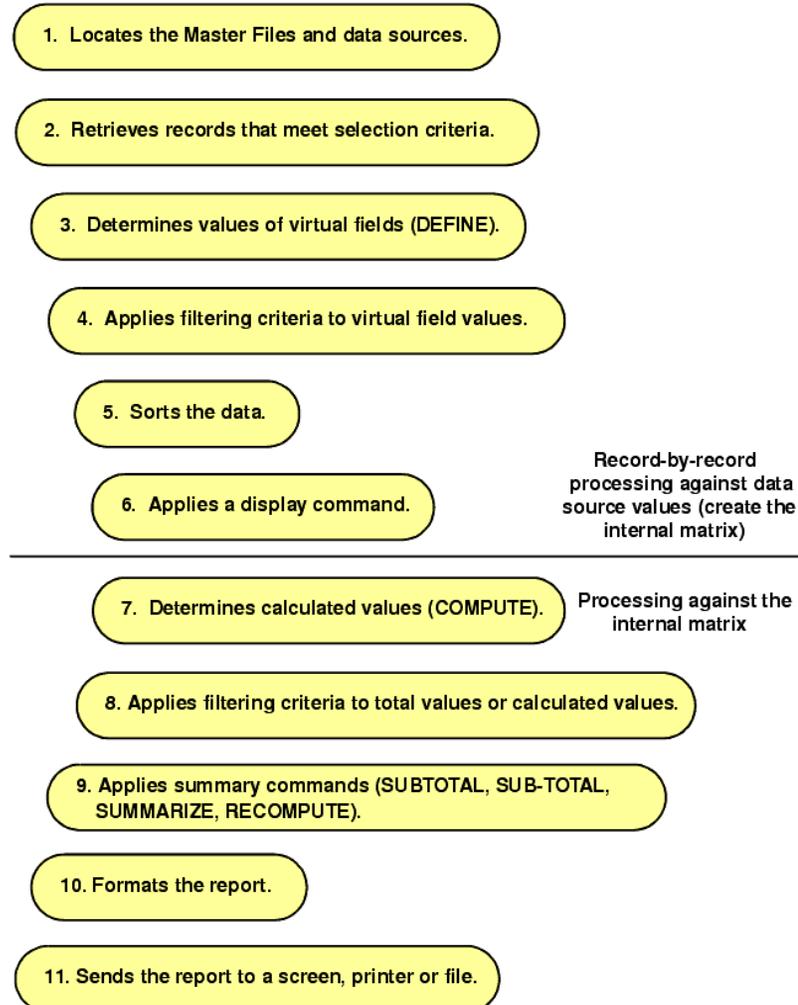
You can use two types of temporary fields (a *virtual field* and a *calculated value*), which differ in how they are evaluated:

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.

A calculated value (COMPUTE) is evaluated after all the data that meets the selection criteria is retrieved, sorted, and summed. Therefore, the calculation is performed using the aggregated values of the fields.

Reference: Evaluation of Temporary Fields

The following illustration shows how a request processes, and when each type of temporary field is evaluated:



Example: Distinguishing Between Virtual Fields and Calculated Values

In the following example, both the DRATIO field (virtual field) and the CRATIO (calculated value) use the same expression DELIVER_AMT/OPENING_AMT, but do not return the same result. The value for CRATIO is calculated after all records have been selected, sorted, and aggregated. The virtual field DRATIO is calculated for each retrieved record.

```
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 output is:

DELIVER_AMT	OPENING_AMT	DRATIO	CRATIO
----- 760	----- 724	----- 28.41	----- 1.05

Reference: Selecting a Temporary Field

The following is to help you choose the kind of temporary field you need.

Choose a virtual field when you want to:

- ❑ Use the temporary field to select data for your report. You cannot use a calculated value, since it is evaluated after data selection takes place.
- ❑ Use the temporary field to sort on data values. A calculated value is evaluated after the data is sorted. With the BY TOTAL phrase, you can sort on this type of field.

Choose a calculated value when you want to:

- ❑ Evaluate the temporary field using total values or prefix operators (which operate on total values). You cannot use a virtual field, since it is evaluated before any totaling takes place.
- ❑ Evaluate the temporary field using fields from different paths in the data structure. You cannot use a virtual field, since it is evaluated before the relationship between data in the different paths is established.

Defining a Virtual Field

In this section:

Defining Multiple Virtual Fields

Displaying Virtual Fields

Clearing a Virtual Field

Establishing a Segment Location for a Virtual Field

Defining Virtual Fields Using a Multi-Path Data Source

Increasing the Speed of Calculations in Virtual Fields

Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN

Applying Dynamically Formatted Virtual Fields to Report Columns

How to:

Create a Virtual Field

Reference:

Usage Notes for Creating Virtual Fields

A virtual field can be used in a request as though it is a real data source field. 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.

You can define a virtual field in the following ways:

- ❑ **In a Master File.** These virtual fields are available whenever the data source is used for reporting. These fields cannot be cleared by JOIN or DEFINE FILE commands.

For more information, see the *Describing Data* manual.

- ❑ **In a procedure.** A virtual field created in a procedure lasts either for the session or until it is cleared by a:
 - ❑ DEFINE FILE *filename* CLEAR command.
 - ❑ Subsequent DEFINE command without the ADD phrase against the same data source.
 - ❑ JOIN command.

Tip: If your environment supports the KEEPDEFINES parameter, you can set KEEPDEFINES to ON to protect virtual fields from being cleared by a subsequent JOIN command.

Reference: Usage Notes for Creating Virtual Fields

- ❑ If you do not use the KEEPDEFINES parameter, when a JOIN is issued, all pre-existing virtual fields for that data source are cleared except those defined in the Master File.
- ❑ To join structures using a virtual field with the source, make sure the DEFINE follows the JOIN command. Otherwise, the JOIN command clears the temporary field. For an explanation of reporting on joined data sources, see [Joining Data Sources](#) on page 831.
- ❑ 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 217.
- ❑ WITH can be used to move the logical home for the virtual field to a segment lower than that to which it would otherwise be assigned (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 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 [Displaying Report Data](#) on page 45.
- ❑ Virtual fields are only available when the data source is used for reporting. Virtual fields cannot be used with MODIFY.
- ❑ A DEFINE command 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.
- ❑ Using a self-referencing DEFINE such as `x=x+1` disables AUTOPATH (see the *Developing Applications* manual).
- ❑ Field names used in the expression that defines the virtual field cannot be enclosed in single or double quotation marks. Any character string enclosed in quotation marks is treated as a literal string, not a field reference.
- ❑ A DEFINE FILE command overwrites a DEFINE in the Master File with same name as long as you do not redefine the format (which is not allowed).

Syntax: How to Create a Virtual Field

Before you begin a report request, include

```
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*.

All 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 [Rotating a Data Structure for Enhanced Retrieval](#) on page 904.

The DEFINE FILE command line must be on a separate line from its virtual field definitions.

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. CLEAR is the default value.

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 210.

REDEFINES *qualifier.fieldname*

Enables you to redefine or recompute a field whose name exists in more than one segment. If you change the format of the field when redefining it, the length in the new format must be the same as or shorter than the original. In addition, conversion between alphanumeric and numeric data types is not supported.

expression

Can be an arithmetic or logical expression or function, evaluated to establish the value of *fieldname* (see [Using Expressions](#) on page 323). You must end each expression with a semicolon except for 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 210.

END

Is required to end the DEFINE FILE command. END must be on its own line in the procedure.

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:

DELIVER_AMT	OPENING_AMT	RATIO
-----	-----	-----
80	65	1.23
100	100	1.00
80	90	.89

Example: Redefining a Field

The following request redefines the salary field in the EMPDATA data source to print asterisks for job titles that contain the word EXECUTIVE:

```
DEFINE FILE EMPDATA
SALARY REDEFINES EMPDATA.SALARY =
  IF TITLE CONTAINS 'EXECUTIVE' THEN 999999999999 ELSE
  EMPDATA.SALARY;
END

TABLE FILE EMPDATA
SUM SALARY BY TITLE
WHERE TITLE CONTAINS 'MANAGER' OR 'MARKETING' OR 'SALES'
ON TABLE SET PAGE OFF
END
```

The output is:

TITLE	SALARY
-----	-----
EXEC MANAGER	\$54,100.00
EXECUTIVE MANAGER	*****
MANAGER	\$270,500.00
MARKETING DIRECTOR	\$176,800.00
MARKETING EXECUTIVE	*****
MARKETING SUPERVISOR	\$50,500.00
SALES EXECUTIVE	*****
SALES MANAGER	\$70,000.00
SALES SPECIALIST	\$82,000.00
SENIOR SALES EXEC.	\$43,400.00

Example: Redefining a Field That Has the Same Name in Multiple Segments

The following request joins the EMPDATA data source to itself. This creates a two-segment structure in which the names are the same in both segments. The request then redefines the salary field in the top segment (tag name ORIG) so that all names starting with the letter L are replaced by asterisks, and redefines the salary field in the child segment (tag name NEW) so that all names starting with the letter M are replaced by asterisks:

```

JOIN PIN IN EMPDATA TAG ORIG TO PIN IN EMPDATA TAG NEW AS AJ
DEFINE FILE EMPDATA
SALARY/D12.2M REDEFINES ORIG.SALARY = IF LASTNAME LIKE 'L%' THEN
                                     999999999999 ELSE ORIG.SALARY;
SALARY/D12.2M REDEFINES NEW.SALARY = IF LASTNAME LIKE 'M%' THEN
                                     999999999999 ELSE NEW.SALARY * 1.2;

END
TABLE FILE EMPDATA
PRINT ORIG.SALARY AS 'ORIGINAL' NEW.SALARY AS 'NEW'
BY LASTNAME
WHERE LASTNAME FROM 'HIRSCHMAN' TO 'OLSON'
ON TABLE SET PAGE NOPAGE
END
    
```

The output is:

LASTNAME	ORIGINAL	NEW
-----	-----	---
HIRSCHMAN	\$62,500.00	\$75,000.00
KASHMAN	\$33,300.00	\$39,960.00
LAISTRA	*****	\$138,000.00
LEWIS	*****	\$60,600.00
LIEBER	*****	\$62,400.00
LOPEZ	*****	\$31,680.00
MARTIN	\$49,000.00	*****
MEDINA	\$39,000.00	*****
MORAN	\$30,800.00	*****
NOZAWA	\$80,500.00	\$96,600.00
OLSON	\$30,500.00	\$36,600.00

Defining Multiple Virtual Fields

How to:

Add a Virtual Field to Existing Virtual Fields

You may wish to have more than one set of virtual fields for 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.

Syntax: How to Add a Virtual Field to Existing Virtual Fields

```
DEFINE FILE filename ADD
```

where:

filename

Is the data source.

Example: Adding 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
```

1. The first DEFINE command creates the TYPE virtual field for the CAR data source. For information about the DECODE function, see the *Using Functions* manual.
2. Two or 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.

Displaying Virtual Fields

How to:

Display Virtual Fields

You can display all virtual fields with the ? DEFINE command.

Syntax: How to Display Virtual Fields

```
? DEFINE
```

For more information, see the *Developing Applications* manual.

Clearing a Virtual Field

The following can clear a virtual field created in a procedure:

- ❑ A DEFINE FILE *filename* CLEAR command.
- ❑ A subsequent DEFINE command (without the ADD option), against the same data source.
- ❑ A join. When a join is created 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.
- ❑ A change in the value of the FIELDNAME SET parameter.

Unlike fields created in a procedure, virtual fields in the Master File are not cleared in the above ways.

Example: Clearing Virtual Fields

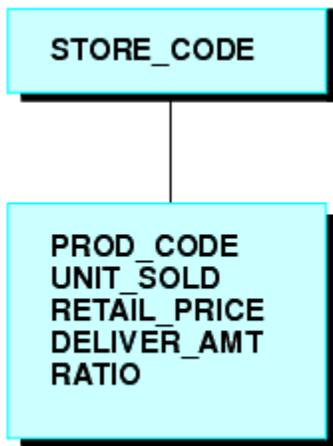
The following annotated example illustrates the use of the CLEAR options for virtual fields:

```
1. DEFINE FILE CAR
   ETYPE/A2=DECODE STANDARD (OHV O OHC O ELSE L);
   END
2. DEFINE FILE CAR CLEAR
   COST = RCOST-DCOST;
   END
```

1. The first DEFINE command creates the TYPE virtual field for the CAR data source. For information about the DECODE function, see the *Using Functions* manual.
2. The CLEAR option clears the 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, and determines the time of execution for that field. Consider the following data source structure and DEFINE command:



```
DEFINE 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 only to 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 positions in the data source cannot be determined. You have to specify in which segment you want the expression 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 FOCUS data sources, 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.

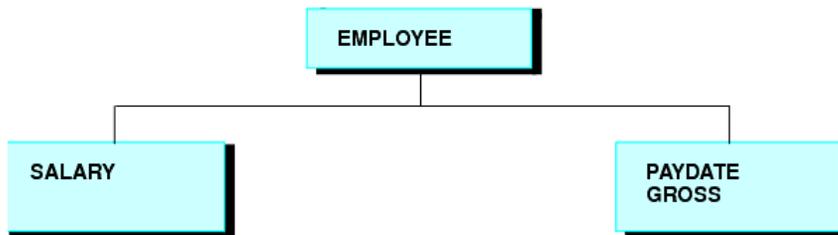
Example: Establishing a Segment Location

The field NCOUNT is placed in the same segment as the UNITS field. NCOUNT is calculated each time a new segment instance is retrieved.

```
DEFINE FILE GGSALES  
NCOUNT/I5 WITH UNITS = NCOUNT+1;  
END
```

Defining Virtual Fields Using a Multi-Path Data Source

Calculations 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 does 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 [Improving Report Processing](#) on page 903.

Increasing the Speed of Calculations in Virtual Fields

Virtual fields can be compiled into machine code in order to increase the speed of calculations. For more information, see Chapter 16, [Improving Report Processing](#).

Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN

How to:

Protect Virtual Fields From Being Overwritten

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. 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. You can return to the default context with the DEFINE FILE RETURN command, and the virtual field definitions remain 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 [Joining Data Sources](#) on page 831.

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 Overwritten**

```
DEFINE FILE filename SAVE  
fld1/format1=expression1;  
fld2/format2=expression2 ;  
END  
TABLE FILE filename ...  
MODIFY FILE filename ...  
DEFINE FILE filename RETURN  
END
```

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.

Applying Dynamically Formatted Virtual Fields to Report Columns

How to:

Define and Apply a Format Field

Reference:

Usage Notes for Field-Based Reformatting

Dynamic formatting enables you to apply different formats to specific data in a column by using a temporary field that contains dynamic data settings.

Before you can format a report column using the dynamic format, you must create the report, then apply the temporary field to a column in the report. For example, you can create a temporary field that contains different decimal currency formats for countries like Japan (which uses no decimal places) and England (which uses 2 decimal places). These currency formats are considered dynamic formats. You can then apply the temporary field containing the dynamic formatting to a Sales column. In a report, the Sales column reflects the different currency formats for each 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 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 of 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.

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

```
TABLE FILE filename  
displayfieldname/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 is used. For example, a field named I8 cannot be used for field-based reformatting, because it is 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: Creating Dynamically Formatted Fields

The following request formats the SALES field according to the value of the COUNTRY field:

```
DEFINE FILE CAR
MYFORMAT/A8=DECODE COUNTRY ('ENGLAND' 'P15.3C' 'JAPAN' 'P15.0' ELSE
'P15.2M');
END

TABLE FILE CAR
SUM SALES/MYFORMAT BY COUNTRY
END
```

The output is:

COUNTRY	SALES
-----	-----
ENGLAND	12,000.000
FRANCE	\$.00
ITALY	\$30,200.00
JAPAN	78030.
W GERMANY	\$88,190.00

Reference: Usage Notes for Field-Based Reformatting

- ❑ Field-based reformatting is supported for TABLE and TABLEF. It works with StyleSheets, joins, and 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 is 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 are 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 are not aligned in the report output.

Creating a Calculated Value

In this section:

Using Positional Column Referencing With Calculated Values

Using ACROSS With Calculated Values

Sorting Calculated Values

Screening on Calculated Values

How to:

Create a Calculated Value

Create a Calculated Value Without a Calculation

Reference:

Usage Notes for Calculated Field Values

A calculated value is a temporary field that is evaluated after all the data that meets the selection criteria is retrieved, sorted, and summed. Calculated values are available only for the specified report request.

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.

Reference: [Usage Notes for Calculated Field Values](#)

The following apply to the use of calculated 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. For information, see the examples in [Including Totals and Subtotals](#) on page 269.
- ❑ Expressions in a COMPUTE command can include fields with prefix operators (see [Manipulating Display Fields With Prefix Operators](#) on page 60). For more information on valid expressions, see [Using Expressions](#) on page 323.
- ❑ Fields referred to in a COMPUTE command are counted toward the display field limit, and appear in the internal matrix. For details on determining the maximum number of display fields that can be used in a request, see [Displaying Report Data](#) on page 45.

- ❑ Field names used in the expression that defines the calculated field cannot be enclosed in single or double quotation marks. Any character string enclosed in quotation marks is treated as a literal string, not a field reference.
- ❑ When using a COMPUTE with an ACROSS COLUMNS phrase, the COLUMNS should be specified last:

```
ACROSS acrossfield [AND] COMPUTE compute_expression; COLUMNS values
```

Syntax: How to Create a Calculated Value

```
COMPUTE fld [/format]= expression;[AS 'title'] [NOPRINT] [IN [+n]]
```

where:

fld

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 (_). It 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 [Using Expressions](#) on page 323). Each field used in the expression must be part of the request. Each expression must end with a semicolon.

NOPRINT

Suppresses printing of the field. .

AS '*title*'

Changes the name of the calculated value. F

IN [+*n*]

Specifies the location of the column.

Syntax: **How to Create a Calculated Value Without a Calculation**

```
COMPUTE fld [/format]= ;
```

where:

fld

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 (_). It 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.

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 this 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  UNITS  RETAIL
CODE  SOLD   PRICE          REVENUE
-----
B10   30     $.85           $25.50
B17   20     $1.89          $37.80
B20   15     $1.99          $29.85
C13   15     $1.99          $29.85
C14   18     $2.05          $36.90
C17   12     $2.09          $25.08
D12   20     $2.09          $41.80
E1    30     $.89           $26.70
E2    33     $.99           $32.67
E3    35     $1.09          $38.15

```

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 you are 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.

For additional information about column reference numbers, see [Assigning Column Reference Numbers](#) on page 229.

Example: Using Positional Column Referencing

The following example demonstrates positional field references in a COMPUTE command:

```

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:

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 ACROSS With Calculated Values

If the COMPUTE command is issued immediately following an ACROSS phrase, only a recap type of the calculation is performed once for all columns. COMPUTE is used as part of a display command, so a new column is calculated for each set of values.

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:

CITY	NEWARK		STAMFORD		UNIONDALE		NEWVAL
NEW YORK	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 ACROSS With Calculated Values

In the following COMPUTE command, C1, C2, C3, C4, C5, and C6 are positional column references, and the COMPUTE command follows the ACROSS phrase. The COMPUTE is performed once for the report, and the results are displayed to the right of all sort groups.

```
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
```

The output is:

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 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 [Sorting and Aggregating Report Columns](#) on page 135.

Screening on Calculated Values

You can screen on values produced by COMPUTE commands by using the WHERE TOTAL test, as described in [Selecting Records for Your Report](#) on page 157.

Assigning Column Reference Numbers

In this section:

Using Column Notation in a Report Request

How to:

Control the Creation of Column Reference Numbers

Column notation assigns a sequential column number to each column in the internal matrix created for a report request. If you want to control the creation of column reference numbers for the columns that are used in your report, use the CNOTATION column notation command.

Because column numbers refer to columns in the internal matrix, they are assigned after retrieval and aggregation of data are completed. Columns created and displayed in a report are stored in the internal matrix, and columns that are not displayed in a report may also be generated and stored in the internal matrix. Columns stored in the internal matrix include calculated values, reformatted field values, BY fields, fields with the NOPRINT option, and certain RECAP calculations such as FORECAST and REGRESS. Every other column in the internal matrix is assigned a column number by default which means you have to account for all internally generated columns if you want to refer to the appropriate column value in your request.

You can change the default assignment of column reference numbers by using the SET CNOTATION command which can assign column numbers only to columns that display in the report output or to all fields referenced in the report request. You can use column notation in COMPUTE and RECAP commands to refer to these columns in your request.

Syntax: **How to Control the Creation of Column Reference Numbers**

```
SET CNOTATION={ALL | PRINTONLY | EXPLICIT}
```

where:

ALL

Assigns column reference numbers to every column in the internal matrix. ALL is the default value.

PRINTONLY

Assigns column reference numbers only to columns that display in the report output.

EXPLICIT

Assigns column reference numbers to all fields referenced in the request, whether displayed or not.

Using Column Notation in a Report Request

Reference:
Usage Notes for Column Numbers

To create a column reference in a request, you can:

- ❑ Preface the column number with a C in a non-FML request.
- ❑ Use the column number as an index in conjunction with a row label in an FML request. With this type of notation, you can specify a specific column, a relative column number, or a sequence or series of columns.
- ❑ Refer to a particular cell in an FML request using the notation E(r,c), where r is a row number and c is a column number.

Example: **Using Column Notation in a Non-FML Request With CNOTATION=ALL**

In the following request with CNOTATION=ALL, the product of C1 and C2 *does not* calculate TRANSTOT times QUANTITY because the reformatting generates additional columns.

```
SET CNOTATION = ALL  
  
TABLE FILE VIDEOTRK  
SUM TRANSTOT/D12.2 QUANTITY/D12.2  
AND COMPUTE  
PRODUCT = C1 * C2;  
BY TRANSDATE  
END
```

The output is:

TRANSDATE	TRANSTOT	QUANTITY	PRODUCT
-----	-----	-----	-----
91/06/17	57.03	12.00	3,252.42
91/06/18	21.25	2.00	451.56
91/06/19	38.17	5.00	1,456.95
91/06/20	14.23	3.00	202.49
91/06/21	44.72	7.00	1,999.88
91/06/24	126.28	12.00	15,946.63
91/06/25	47.74	8.00	2,279.11
91/06/26	40.97	2.00	1,678.54
91/06/27	60.24	9.00	3,628.85
91/06/28	31.00	3.00	961.00

BY fields do not get a column reference, so the first column reference is for TRANSTOT with its original format, then the reformatted version. Next is QUANTITY with its original format and then the reformatted version. Last is the calculated value, PRODUCT.

Example: Using Column Notation in a Non-FML Request With CNOTATION=PRINTONLY

Setting CNOTATION=PRINTONLY assigns column references to the output columns only. In this case, the product of C1 and C2 *does* calculate TRANSTOT times QUANTITY.

```
SET CNOTATION = PRINTONLY

TABLE FILE VIDEOTRK
SUM TRANSTOT/D12.2 QUANTITY/D12.2
AND COMPUTE
PRODUCT = C1 * C2;
BY TRANSDATE
END
```

The output is:

TRANSDATE	TRANSTOT	QUANTITY	PRODUCT
-----	-----	-----	-----
91/06/17	57.03	12.00	684.36
91/06/18	21.25	2.00	42.50
91/06/19	38.17	5.00	190.85
91/06/20	14.23	3.00	42.69
91/06/21	44.72	7.00	313.04
91/06/24	126.28	12.00	1,515.36
91/06/25	47.74	8.00	381.92
91/06/26	40.97	2.00	81.94
91/06/27	60.24	9.00	542.16
91/06/28	31.00	3.00	93.00

Example: Using CNOTATION=PRINTONLY With Column Numbers in an FML Request

In the following request, the reformatting of fields generates additional columns in the internal matrix. In the second RECAP expression, note that because of the CNOTATION setting:

- ❑ TOTCASH(1) refers to total cash in displayed column 1.
- ❑ TOTCASH(2) refers to total cash in displayed column 2.
- ❑ The resulting calculation is displayed in column 2 of the row labeled CASH GROWTH(%).
- ❑ The RECAP value is only calculated for the column specified.

```

SET CNOTATION=PRINTONLY
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
END
TABLE FILE LEDGER
SUM CUR_YR/F9.2 AS 'CURRENT, YEAR'
LAST_YR/F9.2 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/F9.2C= R1 + R2 + R3; AS 'TOTAL CASH' OVER
" " OVER
RECAP GROCASH(2)/F9.2C=100*TOTCASH(1)/TOTCASH(2) - 100;
AS 'CASH GROWTH(%)'
END
    
```

The output is:

	CURRENT YEAR	LAST YEAR
	-----	-----
CASH ON HAND	8784.00	7214.00
DEMAND DEPOSITS	4494.00	3482.00
TIME DEPOSITS	7961.00	6499.00
	-----	-----
TOTAL CASH	21,239.00	17,195.00
CASH GROWTH(%)		23.52

Example: Using CNOTATION=PRINTONLY to RECAP Over Contiguous Columns in an FML Request

In this example, the RECAP calculation for ATOT occurs only for displayed columns 2 and 3, as specified in the request. No calculation is performed for displayed column 1.

```
SET CNOTATION=PRINTONLY
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/F9.2 CUR_YR/F9.2 LAST_YR/F9.2
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	25991.00	21239.00	17195.00
ACCOUNTS RECEIVABLE	21941.00	18829.00	15954.00
INVENTORY	31522.00	27307.00	23329.00
	-----	-----	-----
ASSETS ACTUAL		67,375	56,478

Example: Using CNOTATION=PRINTONLY With Relative Column Addressing in an FML Request

This example computes the change in cash (CHGCASH) for displayed columns 1 and 2.

```
SET CNOTATION=PRINTONLY
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/F9.2 CUR_YR/F9.2 LAST_YR/F9.2
FOR ACCOUNT
10$$ AS 'TOTAL CASH' LABEL TOTCASH            OVER
" "                                           OVER
RECAP CHGCASH(1,2)/I5C = TOTCASH(*) - TOTCASH(*+1); AS 'CHANGE IN CASH'
END
```

The output is:

	NEXT_YR -----	CUR_YR -----	LAST_YR -----
TOTAL CASH	25991.00	21239.00	17195.00
CHANGE IN CASH	4,752	4,044	

Example: Using CNOTATION=PRINTONLY With Cell Notation in an FML Request

In this request, two RECAP expressions derive VARIANCEs (EVAR and WVAR) by subtracting values in four displayed columns (1, 2, 3, 4) in row three (PROFIT); these values are identified using cell notation (r,c).

```

SET CNOTATION=PRINTONLY
TABLE FILE REGION
SUM E_ACTUAL/F9.2 E_BUDGET/F9.2 W_ACTUAL/F9.2 W_BUDGET/F9.2
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	6000.00	4934.00	7222.00	7056.00
COST	4650.00	3760.00	5697.00	5410.00
PROFIT	1,350	1,174	1,525	1,646
EAST VARIANCE	176			
WEST VARIANCE			-121	

Example: Using NOPRINT, Field Reformatting, and COMPUTE With Column Notation

The following request has a field that is not printed, several reformatted fields and three calculated values. With SET CNOTATION=PRINONLY, the column references result in correct output.

```

SET CNOTATION = PRINONLY
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 NOPRINT CUR_YR
COMPUTE AMT2/D6 = AMOUNT *2;
LAST_YR/D5     AMOUNT NEXT_YR
COMPUTE AMT3/D6 = AMOUNT*3;
COMPUTE AMT4/D6 = AMOUNT*4;
FOR ACCOUNT
10$$ AS 'CASH'                                OVER
1100 AS 'ACCTS. REC.'                          OVER
1200 AS 'INVENTORY'                            OVER
BAR                                             OVER
RECAP ATOT/I8C = R1 + R2 + R3; AS 'TOTAL'      OVER
RECAP DIFF(2,10,2)/D8 = ATOT(*) - ATOT(*-1);
END

```

The output is:

	CUR_YR	AMT2	LAST_YR	AMOUNT	NEXT_YR	AMT3	AMT4
	-----	-----	-----	-----	-----	-----	-----
CASH	21,239	42,478	17,195	21,239	25,991	63,717	84,956
ACCTS. REC.	18,829	37,658	15,954	18,829	21,941	56,487	75,316
INVENTORY	27,307	54,614	23,329	27,307	31,522	81,921	109,228
	-----	-----	-----	-----	-----	-----	-----
TOTAL	67,375	134,750	56,478	67,375	79,454	202,125	269,500
DIFF		67,375		10,897		122,671	

Example: Using Column Notation With NOPRINT in a non-FML Request

The following request, sums TRANSTOT, QUANTITY, and TRANSCODE by TRANSDATE. TRANSTOT has the NOPRINT option, so it is not displayed on the report output. The request also calculates the following fields using COMPUTE commands:

- ❑ TTOT2, which has the same value as TRANSTOT and displays on the report output.
- ❑ UNIT_COST1, which is calculated by dividing column1 by column2.

- ❑ UNIT_COST2, which is calculated by dividing column1 by QUANTITY.

```

SET CNOTATION = ALL
TABLE FILE VIDEOTRK
SUM TRANSTOT/D7.2 NOPRINT QUANTITY/D7.2 TRANSCODE
  COMPUTE TTOT2/D7.2 = C1;
  COMPUTE UNIT_COST1/D7.2 = C1/C2;
  COMPUTE UNIT_COST2/D7.2 = C1/QUANTITY;
BY TRANSDATE
END
    
```

With this request, only CNOTATION=EXPLICIT produces the correct output. The following discussion illustrates why the EXPLICIT setting is needed.

With CNOTATION=ALL, all fields in the internal matrix are assigned column numbers. In particular, the request creates the following column references:

- ❑ C1 is TRANSTOT with its original format.
- ❑ C2 is TRANSTOT with format D7.2.
- ❑ C3 is QUANTITY with its original format.
- ❑ C4 is QUANTITY with format D7.2.
- ❑ C5 is TRANSCODE.

UNIT_COST1 is C1/C2. These column numbers have both been assigned to TRANSTOT, so UNIT_COST1 always equals 1. UNIT_COST2 is C1 (TRANSTOT) divided by QUANTITY. The output is:

TRANSDATE	QUANTITY	TRANSCODE	TTOT2	UNIT_COST1	UNIT_COST2
91/06/17	12.00	10	57.03	1.00	4.75
91/06/18	2.00	2	21.25	1.00	10.63
91/06/19	5.00	4	38.17	1.00	7.63
91/06/20	3.00	3	14.23	1.00	4.74
91/06/21	7.00	6	44.72	1.00	6.39
91/06/24	12.00	9	126.28	1.00	10.52
91/06/25	8.00	7	47.74	1.00	5.97
91/06/26	2.00	2	40.97	1.00	20.48
91/06/27	9.00	7	60.24	1.00	6.69
91/06/28	3.00	3	31.00	1.00	10.33

With CNOTATION = PRINTONLY, the field TRANSTOT, which has the NOPRINT option, is not assigned any column numbers. QUANTITY with its original format is not assigned a column number because it is not displayed on the report output. The reformatted QUANTITY field is displayed and is assigned a column number. Therefore, the request creates the following column references:

- ❑ C1 is QUANTITY with format D7.2.
- ❑ C2 is TRANSCODE.

UNIT_COST1 is C1/C2, QUANTITY/TRANSCODE. UNIT_COST2 is C1 (QUANTITY) divided by QUANTITY. Therefore, UNIT_COST2 always equals 1. The output is:

TRANSDATE	QUANTITY	TRANSCODE	TTOT2	UNIT_COST1	UNIT_COST2
91/06/17	12.00	10	12.00	1.20	1.00
91/06/18	2.00	2	2.00	1.00	1.00
91/06/19	5.00	4	5.00	1.25	1.00
91/06/20	3.00	3	3.00	1.00	1.00
91/06/21	7.00	6	7.00	1.17	1.00
91/06/24	12.00	9	12.00	1.33	1.00
91/06/25	8.00	7	8.00	1.14	1.00
91/06/26	2.00	2	2.00	1.00	1.00
91/06/27	9.00	7	9.00	1.29	1.00
91/06/28	3.00	3	3.00	1.00	1.00

With CNOTATION = EXPLICIT, the reformatted TRANSTOT field is explicitly referenced in the request, so it is assigned a column number even though it is not displayed. However, the TRANSTOT field with its original format is not assigned a column number. The QUANTITY field with its original format is not assigned a column number because it is not explicitly referenced in the request. The reformatted QUANTITY field is assigned a column number. Therefore, the request creates the following column references:

- ❑ C1 is TRANSTOT with format D7.2.
- ❑ C2 is QUANTITY with format D7.2.
- ❑ C3 is TRANSCODE.

UNIT_COST1 is C1/C2, TRANSTOT/QUANTITY. UNIT_COST2 is C1 (TRANSTOT) divided by QUANTITY. Therefore, UNIT_COST2 always equals UNIT_COST1. The output is:

TRANSDATE	QUANTITY	TRANSCODE	TTOT2	UNIT_COST1	UNIT_COST2
91/06/17	12.00	10	57.03	4.75	4.75
91/06/18	2.00	2	21.25	10.63	10.63
91/06/19	5.00	4	38.17	7.63	7.63
91/06/20	3.00	3	14.23	4.74	4.74
91/06/21	7.00	6	44.72	6.39	6.39
91/06/24	12.00	9	126.28	10.52	10.52
91/06/25	8.00	7	47.74	5.97	5.97
91/06/26	2.00	2	40.97	20.48	20.48
91/06/27	9.00	7	60.24	6.69	6.69
91/06/28	3.00	3	31.00	10.33	10.33

Example: Using Cell Notation in an FML Request

In the following request, CUR_YR has the NOPRINT option. The CHGCASH RECAP expression is supposed to subtract CUR_YR from LAST_YR and NEXT_YR.

```

SET CNOTATION = ALL
DEFINE FILE LEDGER
CUR_YR/I7C = AMOUNT;
LAST_YR/I5C = .87*CUR_YR - 142;
NEXT_YR/I5C = 1.13*CUR_YR + 222;
END
TABLE FILE LEDGER
SUM CUR_YR/I5C NOPRINT LAST_YR NEXT_YR
FOR ACCOUNT
10$$ AS 'TOTAL CASH ' LABEL TOTCASH OVER
" " OVER
RECAP CHGCASH(1,3)/I5SC=(TOTCASH(*) - TOTCASH(1));
AS 'CHANGE FROM CURRENT'
END
    
```

When CNOTATION = ALL, C1 refers to the CUR_YR field with its original format, C2 refers to the reformatted value, C3 is LAST_YR, and C4 is NEXT_YR. Since there is an extra column and the RECAP only refers to columns 1 and 3, the calculation for NEXT_YR - CUR_YR is not performed. The output is:

	LAST_YR -----	NEXT_YR -----
TOTAL CASH	17,195	25,991
CHANGE FROM CURRENT	-4,044	

When CNOTATION = PRINTONLY, the CUR_YR field is not assigned any column number, so there is no column 3. Therefore, no calculations are performed. The output is:

	LAST_YR -----	NEXT_YR -----
TOTAL CASH	17,195	25,991
CHANGE FROM CURRENT		

When CNOTATION = EXPLICIT, the reformatted version of the CUR_YR field is C1 because it is referenced in the request even though it is not displayed. Both calculations are performed correctly. The output is:

	LAST_YR -----	NEXT_YR -----
TOTAL CASH	17,195	25,991
CHANGE FROM CURRENT	-4,044	4,752

Reference: Usage Notes for Column Numbers

- ❑ BY fields are not assigned column numbers.

- ❑ ACROSS columns are assigned column numbers.
- ❑ Calculated fields are assigned column numbers.
- ❑ Column numbers outside the range of the columns created in the request are allowed under the following circumstances (and are treated as containing the value zero):
 - ❑ When specified in a COMPUTE command issued after an ACROSS phrase.
 - ❑ In a cell reference in an FML RECAP command.

In those cases, it is not possible to know in advance how many columns will be generated by the syntax. Using a column number outside of the range in any other context generates the following message:

```
(FOC258) FIELDNAME OR COMPUTATIONAL ELEMENT NOT RECOGNIZED: column
```

Calculating Trends and Predicting Values With FORECAST

In this section:

FORECAST Processing

Using a Simple Moving Average

Using Single Exponential Smoothing

Using Double Exponential Smoothing

Using Triple Exponential Smoothing

Using a Linear Regression Equation

FORECAST Reporting Techniques

You can calculate trends in numeric data and predict values beyond the range of those stored in the data source by using the FORECAST feature. FORECAST can be used in a report or graph request.

The calculations you can make to identify trends and forecast values are:

- ❑ **Simple moving average (MOVAVE)** calculates a series of arithmetic means using a specified number of values from a field. For details, see [Using a Simple Moving Average](#) on page 245.
- ❑ **Exponential moving average** calculates a weighted average between the previously calculated value of the average and the next data point. There are three methods for using an exponential moving average:

- ❑ **Single exponential smoothing (EXPAVE)** calculates an average that allows you to choose weights to apply to newer and older values. For details, see [Using Single Exponential Smoothing](#) on page 248.
- ❑ **Double exponential smoothing (DOUBLEXP)** accounts for the tendency of data to either increase or decrease over time without repeating. For details, see [Using Double Exponential Smoothing](#) on page 251.
- ❑ **Triple exponential smoothing (SEASONAL)** accounts for the tendency of data to repeat itself in intervals over time. For details, see [Using Triple Exponential Smoothing](#) on page 252.
- ❑ **Linear regression analysis (REGRESS)** derives the coefficients of a straight line that best fits the data points and uses this linear equation to estimate values. For details, see [Usage Notes for Creating Virtual Fields](#) on page 210.

When predicting values in addition to calculating trends, FORECAST continues the same calculations beyond the data points by using the generated trend values as new data points. For the linear regression technique, the calculated regression equation is used to derive trend and predicted values.

FORECAST performs the calculations based on the data provided, but decisions about their use and reliability are the responsibility of the user. Therefore, FORECAST predictions are not always reliable, and many factors determine how accurate a prediction will be.

You can conditionally format forecasted values. For details, see [Styling Reports](#) on page 491.

FORECAST Processing

How to:

Calculate Trends and Predict Values

Reference:

Usage Notes for FORECAST

FORECAST Limits

You invoke FORECAST processing by including FORECAST in a RECAP command. In this command, you specify the parameters needed for generating estimated values, including the field to use in the calculations, the type of calculation to use, and the number of predictions to generate. The RECAP field that contains the result of FORECAST can be a new field (non-recursive) or the same field used in the FORECAST calculations (recursive):

- ❑ In a *recursive* FORECAST, the RECAP field that contains the results is also the field used to generate the FORECAST calculations. 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 is stored in an output file unless you set HOLDLIST to PRINTONLY.
- ❑ In a *non-recursive* FORECAST, a new field contains the results of FORECAST calculations. The new field is displayed in the report along with the original field when it is referenced in the display command. The new field contains trend values and forecast values when specified.

FORECAST operates on the last ACROSS field in the request. If the request does not contain an ACROSS field, it operates on the last BY field. 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. When using an ACROSS field with FORECAST, the display command must be SUM or COUNT.

Note: Although you pass parameters to FORECAST using an argument list in parentheses, FORECAST is not a function. It can coexist with a function of the same name, as long as the function is not specified in a RECAP command.

Syntax: How to Calculate Trends and Predict Values

MOVAVE calculation

```
ON sortfield RECAP result_field[/fmt] = FORECAST(infield, interval,
npredict, 'MOVAVE',npoint1)sendstyle
```

EXPAVE calculation

```
ON sortfield RECAP result_field[/fmt] = FORECAST(infield, interval,  
npredict, 'EXPAVE',npoint1);
```

DOUBLEXP calculation

```
ON sortfield RECAP fld1[/fmt] = FORECAST(infield,  
interval, npredict, 'DOUBLEXP',npoint1, npoint2);
```

SEASONAL calculation

```
ON sortfield RECAP fld1[/fmt] = FORECAST(infield,  
interval, npredict, 'SEASONAL', nperiod, npoint1, npoint2, npoint3);
```

REGRESS calculation

```
ON sortfield RECAP result_field[/fmt] = FORECAST(infield, interval,  
npredict, 'REGRESS');
```

where:

sortfield

Is the last ACROSS field in the request. This field must be in numeric or date format. If the request does not contain an ACROSS field, FORECAST works on the last BY field.

result_field

Is the field containing the result of FORECAST. It can be a new field, or the same as *infield*. This must be a numeric field; either 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 *sortfield*=.

fmt

Is the display format for *result_field*. The default format is D12.2. If *result_field* was previously reformatted using a DEFINE or COMPUTE command, the format specified in the RECAP command is respected.

infield

Is any numeric field. It can be the same field as *result_field*, or a different field. It cannot be a date-time field or a numeric field with date display options.

interval

Is the increment to add to each *sortfield* value (after the last data point) to create the next value. This must be a positive integer. To sort in descending order, use the BY HIGHEST phrase. The result of adding this number to the *sortfield* values is converted to the same format as *sortfield*.

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 an integer greater than or equal to zero. Zero indicates that you do not want predictions, and is only supported with a non-recursive FORECAST. For the SEASONAL method, *npredict* is the number of *periods* to calculate. The number of *points* generated is:

$$nperiod * npredict$$

nperiod

For the SEASONAL method, is a positive whole number that specifies the number of data points in a period.

npoint1

Is the number of values to average for the MOVAVE method. For EXPAVE, DOUBLEXP, and SEASONAL, this number is used to calculate the weights for each component in the average. This value must be a positive whole number. The weight, *k*, is calculated by the following formula:

$$k=2/(1+npoint1)$$

npoint2

For DOUBLEXP and SEASONAL, this positive whole number is used to calculate the weights for each term in the trend. The weight, *g*, is calculated by the following formula:

$$g=2/(1+npoint2)$$

npoint3

For SEASONAL, this positive whole number is used to calculate the weights for each term in the seasonal adjustment. The weight, *p*, is calculated by the following formula:

$$p=2/(1+npoint3)$$

Reference: Usage Notes for FORECAST

- ❑ The sort field used for FORECAST must be in a numeric or date format.

- ❑ When using simple moving average and exponential moving average methods, data values should be spaced evenly in order to get meaningful results.
- ❑ When using a RECAP command with FORECAST, the command can contain only the FORECAST syntax. FORECAST does not recognize any syntax after the closing semicolon (;). To specify options such as AS or IN:
 - ❑ In a *non-recursive* FORECAST request, use an empty COMPUTE command prior to the RECAP.
 - ❑ In a *recursive* FORECAST request, specify the options when the field is first referenced in the report request.
- ❑ The use of column notation is not supported in a request that includes 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.
- ❑ 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 creating a currency-denominated field.
- ❑ Recursive and non-recursive REGRESS are not supported in the same request when the display command is SUM, ADD, or WRITE.
- ❑ Missing values are not supported with REGRESS.
- ❑ If you use the ESTRECORDS parameter to enable the external sort to estimate better the amount of sort work space needed, you must take into account that FORECAST with predictions creates additional records in the output.
- ❑ In a styled report, you can assign specific attributes to values predicted by FORECAST with the StyleSheet attribute WHEN=FORECAST. For example, to make the predicted values display with the color red, use the following syntax in the TABLE request:

```
ON TABLE SET STYLE  
*TYPE=DATA , COLUMN=MYFORECASTSORTFIELD , WHEN=FORECAST , COLOR=RED ,  
$ENDSTYLE
```

Reference: FORECAST Limits

The following are not supported with a RECAP command that uses FORECAST:

- ❑ BY TOTAL command.
- ❑ MORE, MATCH, FOR, and OVER phrases.

- ❑ SUMMARIZE and RECOMPUTE are not supported for the same sort field used for FORECAST.
- ❑ MISSING attribute.

Using a Simple Moving Average

A simple moving average is a series of arithmetic means calculated with a specified number of values from a field. 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 a specified number of periods of stock prices. A disadvantage to this indicator is that because it drops the oldest values from the calculation as it moves on, it loses its memory over time. Also, mean values are distorted by extreme highs and lows, since this method gives 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	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	730317	714,029.7
	10	57012	724412	719,220.8
	11	51110	620264	669,742.4
	12	58981	762328	716,035.2
	13	0	0	739,181.6
	14	0	0	750,754.8
	15	0	0	756,541.4
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	710138	708,167.6
	12	57290	705315	706,741.3
	13	0	0	706,028.2
	14	0	0	705,671.6
	15	0	0	705,493.3

In the report, the number of values to use in the average is 3 and there are no UNITS or DOLLARS values for the generated PERIOD values.

Each average (MOVAVE value) is computed using DOLLARS values where they exist. The calculation of the moving average begins in the following way:

- ❑ 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: $(801,123 + 682,340) / 2$.
- ❑ The third MOVAVE value (749,513.7) is the mean of DOLLARS values one through three: $(801,123 + 682,340 + 765,078) / 3$.
- ❑ The fourth MOVAVE value (712,897.3) is the mean of DOLLARS values two through four: $(682,340 + 765,078 + 691,274) / 3$.

For predicted values beyond the supplied values, the calculated MOVAVE values are used as new data points to continue the moving average. 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 762,328) and the moving average for period 12 (702,334.7). The calculation is: $694,975 = (620,264 + 762,328 + 702,334.7)/3$.

Example: Using an Existing Field as a 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. It 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	730,317.0
	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	710,138.0
	12	57290	705,315.0
	13	0	708,397.8
	14	0	707,817.7
	15	0	708,651.9

Using Single Exponential Smoothing

The single exponential smoothing method calculates an average that allows you to choose weights to apply to newer and older values.

The following formula determines the weight given to the newest value.

$$k = 2 / (1+n)$$

where:

k

Is the newest value.

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 data value in the sort group is used as the next data value.

Example: Calculating a Single Exponential Smoothing Column

The following 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 retrieved data.

```
DEFINE FILE GGSALES
  SDATE/YM = 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	730317	714,029.7
	10	57012	724412	719,220.8
	11	51110	620264	669,742.4
	12	58981	762328	716,035.2
	13	0	0	739,181.6
	14	0	0	750,754.8
	15	0	0	756,541.4
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	710138	708,167.6
	12	57290	705315	706,741.3
	13	0	0	706,028.2
	14	0	0	705,671.6
	15	0	0	705,493.3

In the report, 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.

Each average (EXPAVE value) is computed using DOLLARS values where they exist. The calculation of the moving average begins in the following way:

- ❑ 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)

$$k = 2/(1+n) = 2/4 = 0.5$$

$$\begin{aligned} \text{EXPAVE} &= (\text{EXPAVE} \cdot (1-k)) + (\text{new-DOLLARS} \cdot k) = (801123 \cdot 0.5) + (682340 \cdot 0.5) \\ &= 400561.5 + 341170 = 741731.5 \end{aligned}$$

- The third EXPAVE value (753,404.8) is calculated as follows:

$$\begin{aligned} \text{EXPAVE} &= (\text{EXPAVE} * (1-k)) + (\text{new-DOLLARS} * k) = (741731.5 * 0.5) + (765078 * 0.5) \\ &= 370865.75 + 382539 = 753404.75 \end{aligned}$$

For predicted values beyond those supplied, the last EXPAVE value is used as the new data point in the exponential smoothing calculation. The predicted EXPAVE values (starting with 706,741.6) are 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, the previous average for period 13 is 706,741.6, and this is also used as the next data point. Therefore, the average is calculated as follows: $(706,741.6 * 0.5) + (706,741.6 * 0.5) = 706,741.6$

$$\begin{aligned} \text{EXPAVE} &= (\text{EXPAVE} * (1-k)) + (\text{new-DOLLARS} * k) = (706741.6 * 0.5) + \\ &(706741.6 * 0.5) = 353370.8 + 353370.8 = 706741.6 \end{aligned}$$

Using Double Exponential Smoothing

Double exponential smoothing produces an exponential moving average that takes into account the tendency of data to either increase or decrease over time without repeating. This is accomplished by using two equations with two constants:

- The first equation accounts for the current time period and is a weighted average of the current data value and the prior average, with an added component (b) that represents the trend for the previous period. The weight constant is k:

$$\text{DOUBLEXP}(t) = k * \text{datavalue}(t) + (1-k) * ((\text{DOUBLEXP}(t-1)) + b(t-1))$$

- The second equation is the calculated trend value, and is a weighted average of the difference between the current and previous average and the trend for the previous time period. b(t) represents the average trend. The weight constant is g:

$$b(t) = g * (\text{DOUBLEXP}(t) - \text{DOUBLEXP}(t-1)) + (1 - g) * (b(t-1))$$

These two equations are solved to derive the smoothed average. The first smoothed average is set to the first data value. The first trend component is set to zero. For choosing the two constants, the best results are usually obtained by minimizing the mean-squared error (MSE) between the data values and the calculated averages. You may need to use nonlinear optimization techniques to find the optimal constants.

The equation used for forecasting beyond the data points with double exponential smoothing is

$$\text{forecast}(t+m) = \text{DOUBLEXP}(t) + m * b(t)$$

where:

m

Is the number of time periods ahead for the forecast.

Example: Calculating a Double Exponential Smoothing Column

The following sums the ACTUAL_YTD field of the CENTSTMT data source by period, and calculates a single exponential and double exponential moving average.

```
TABLE FILE CENTSTMT
SUM ACTUAL_YTD
  BY PERIOD
  ON PERIOD RECAP EXP/D15.1 = FORECAST(ACTUAL_YTD,1,0,'EXPAVE',3);
  ON PERIOD RECAP DOUBLEXP/D15.1 = FORECAST(ACTUAL_YTD,1,0,
'DOUBLEXP',3,3);
WHERE GL_ACCOUNT LIKE '3%%%'
END
```

The output is:

PERIOD	YTD Actual	EXP	DOUBLEXP
2002/01	12,957,681.	12,957,681.0	12,957,681.0
2002/02	25,441,971.	19,199,826.0	22,439,246.3
2002/03	39,164,321.	29,182,073.5	34,791,885.1
2002/04	52,733,326.	40,957,699.8	48,845,816.0
2002/05	66,765,920.	53,861,809.9	63,860,955.9
2002/06	80,952,492.	67,407,150.9	79,188,052.9

Using Triple Exponential Smoothing

Triple exponential smoothing produces an exponential moving average that takes into account the tendency of data to repeat itself in intervals over time. For example, sales data that is growing and in which 25% of sales always occur during December contains both trend and seasonality. Triple exponential smoothing takes both the trend and seasonality into account by using three equations with three constants.

For triple exponential smoothing you, need to know the number of data points in each time period (designated as L in the following equations). To account for the seasonality, a seasonal index is calculated. The data is divided by the prior season's index and then used in calculating the smoothed average.

- The first equation accounts for the current time period, and is a weighted average of the current data value divided by the seasonal factor and the prior average adjusted for the trend for the previous period. The weight constant is k:

$$SEASONAL(t) = k * (datavalue(t)/I(t-L)) + (1-k) * (SEASONAL(t-1) + b(t-1))$$

- The second equation is the calculated trend value, and is a weighted average of the difference between the current and previous average and the trend for the previous time period. $b(t)$ represents the average trend. The weight constant is g :

$$b(t) = g * (SEASONAL(t) - SEASONAL(t-1)) + (1-g) * (b(t-1))$$

- The third equation is the calculated seasonal index, and is a weighted average of the current data value divided by the current average and the seasonal index for the previous season. $I(t)$ represents the average seasonal coefficient. The weight constant is p :

$$I(t) = p * (datavalue(t)/SEASONAL(t)) + (1 - p) * I(t-L)$$

These equations are solved to derive the triple smoothed average. The first smoothed average is set to the first data value. Initial values for the seasonality factors are calculated based on the maximum number of full periods of data in the data source, while the initial trend is calculated based on two periods of data. These values are calculated with the following steps:

1. The initial trend factor is calculated by the following formula:

$$b(0) = (1/L) ((y(L+1)-y(1))/L + (y(L+2)-y(2))/L + \dots + (y(2L) - y(L))/L)$$

2. The calculation of the initial seasonality factor is based on the average of the data values within each period, $A(j)$ ($1 \leq j \leq N$):

$$A(j) = (y((j-1)L+1) + y((j-1)L+2) + \dots + y(jL)) / L$$

3. Then, the initial periodicity factor is given by the following formula, where N is the number of full periods available in the data, L is the number of points per period and n is a point within the period ($1 \leq n \leq L$):

$$I(n) = (y(n)/A(1) + y(L+n)/A(2) + \dots + y((N-1)L+n)/A(N)) / N$$

The three constants must be chosen carefully. The best results are usually obtained by choosing the constants to minimize the mean-squared error (MSE) between the data values and the calculated averages. Varying the values of $npoint1$ and $npoint2$ affect the results, and some values may produce a better approximation. To search for a better approximation, you may want to find values that minimize the MSE.

The equation used to forecast beyond the last data point with triple exponential smoothing is:

$$forecast(t+m) = (SEASONAL(t) + m * b(t)) / I(t-L+MOD(m/L))$$

where:

m

Is the number of periods ahead for the forecast.

Example: Calculating a Triple Exponential Smoothing Column

In the following, the data has seasonality but no trend. Therefore, *npoint2* is set high (1000) to make the trend factor negligible in the calculation:

```
SET HISTOGRAM = OFF
TABLE FILE VIDEOTRK
SUM TRANSTOT
BY TRANSDATE
ON TRANSDATE RECAP SEASONAL/D10.1 = FORECAST(TRANSTOT,1,3,'SEASONAL',
3,3,1000,1);
WHERE TRANSDATE NE '19910617'
END
```

In the output, *npredict* is 3. Therefore, three periods (nine points, *nperiod* * *npredict*) are generated.

TRANSDATE	TRANSTOT	SEASONAL
91/06/18	21.25	21.3
91/06/19	38.17	31.0
91/06/20	14.23	34.6
91/06/21	44.72	53.2
91/06/24	126.28	75.3
91/06/25	47.74	82.7
91/06/26	40.97	73.7
91/06/27	60.24	62.9
91/06/28	31.00	66.3
91/06/29		45.7
91/06/30		94.1
91/07/01		53.4
91/07/02		72.3
91/07/03		140.0
91/07/04		75.8
91/07/05		98.9
91/07/06		185.8
91/07/07		98.2

Using a Linear Regression Equation

The Linear Regression Equation estimates values by assuming that the dependent variable (the new calculated values) and the independent variable (the sort field values) are related by a function that represents a straight line:

$$y = mx + b$$

where:

y
Is the dependent variable.

x
Is the independent variable.

m

Is the slope of the line.

b

Is 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.

$$m = \frac{(\sum xy - (\sum x \cdot \sum y)/n)}{(\sum x^2 - (\sum x)^2/n)}$$

$$b = (\sum y)/n - (m \cdot (\sum x)/n)$$

where:

n

Is the number of data points.

y

Are the data values (dependent variables).

x

Are the sort field values (independent variables).

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 calculation process):

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, all FORECAST requests must specify the same sort field, interval, and number of predictions. The only things that can change are the RECAP field, method, field used to calculate the FORECAST values, and number of points to average. 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 a constant other than zero. Rows in the report output that represent actual records in the data source will appear with 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	773136	736,718.0	738,823.2	751,907.0	757,995.6
8	655896	685170	715,202.0	705,652.3	733,630.7	728,865.3
9	730317	753760	711,155.3	715,518.2	737,355.3	738,823.2
10	724412	709397	703,541.7	719,075.7	716,109.0	727,052.7
11	620264	630452	691,664.3	679,551.0	697,869.7	688,412.4
12	762328	718837	702,334.7	712,661.8	686,228.7	700,582.3

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 of the RECAP command takes precedence.

```

DEFINE FILE GGSales
  SDATE/YM = 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	801,123.0	801123
	2	54870	741,731.5	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	711,155.3	730317
	10	57012	703,541.7	724412
	11	51110	691,664.3	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

Calculating Trends and Predicting Values With Multivariate REGRESS

How to:

Create a Multivariate Linear Regression Column

Reference:

Usage Notes for REGRESS

The REGRESS method derives a linear equation that best fits a set of numeric data points, and uses this equation to create a new column in the report output. The equation can be based on one to three independent variables.

This method estimates values by assuming that the dependent variable (y , the new calculated values) and the independent variables (x_1 , x_2 , x_3) are related by the following linear equation:

$$y = a_1 * x_1 [+ a_2 * x_2 [+ a_3 * x_3]] + b$$

When there is one independent variable, the equation represents a straight line. This produces the same values as FORECAST using the REGRESS method. When there are two independent variables, the equation represents a plane, and with three independent variables, it represents a hyperplane. You should use this technique when you have reason to believe that the dependent variable can be approximated by a linear combination of the independent variables.

REGRESS uses a technique called Ordinary Least Squares to calculate values for the coefficients (a_1 , a_2 , a_3 , and b) that minimize the sum of the squared differences between the data and the resulting line, plane, or hyperplane.

Syntax: How to Create a Multivariate Linear Regression Column

```
ON {sortfield} RECAP y[/fmt] = REGRESS(n, x1, [x2, [x3,]] z);
```

where:

sortfield

Is a field in the data source. It cannot be the same field as any of the parameters to REGRESS. A new linear regression equation is derived each time the sort field value changes.

y

Is the new numeric column calculated by applying the regression equation. You cannot DEFINE or COMPUTE a field with this name.

fmt

Is the display format for y . If it is omitted, the default format is D12.2.

n

Is a whole number from 1 to 3 indicating the number of independent variables.

x1, x2, x3

Are the field names to be used as the independent variables. All of these variables must be numeric and be independent of each other.

z

Is an existing numeric field that is assumed to be approximately linearly dependent on the independent variables and is used to derive the regression equation.

Reference: Usage Notes for REGRESS

- ❑ The (By) sort field used with REGRESS must be in a numeric or date format.
- ❑ REGRESS cannot operate on an ACROSS field.
- ❑ If any of the independent variables are also sort fields, they cannot be referenced in the request prior to the REGRESS sort field.
- ❑ FORECAST and REGRESS cannot be used in the same request, and only one REGRESS is supported in a request. Non-REGRESS RECAP commands are supported.
- ❑ The RECAP command used with REGRESS can contain only the REGRESS syntax. REGRESS does not recognize any syntax after the closing semicolon (;).
- ❑ Although you pass parameters to REGRESS using an argument list in parentheses, REGRESS 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.
- ❑ BY TOTAL is not supported.
- ❑ MORE, MATCH, FOR, and OVER are not supported.
- ❑ The process of generating the REGRESS 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 REGRESS.
- ❑ SUMMARIZE and RECOMPUTE are not supported for the same sort field used for REGRESS.
- ❑ REGRESS is not supported for the FOCUS GRAPH facility.
- ❑ The left side of a RECAP command used for REGRESS supports the CURR attribute for creating a currency-denominated field.
- ❑ Fields with missing values cannot be used in the regression.

- ❑ Larger amounts of data produce more useful results.

Example: Creating a Multivariate Linear Regression Column

The following request uses the GGSales data source to calculate an estimated DOLLARS column. The BUDUNITS, UNITS, and BUDDOLLARS fields are the independent variables. The DOLLARS field provides the actual values to be estimated:

```
DEFINE FILE GGSales
  YEAR/Y = DATE;
  MONTH/M = DATE;
  PERIOD/I2 = MONTH;
END

TABLE FILE GGSales
PRINT BUDUNITS UNITS BUDDOLLARS DOLLARS
BY PERIOD
ON PERIOD
RECAP EST_DOLLARS/F8 = REGRESS(3, BUDUNITS, UNITS, BUDDOLLARS, DOLLARS);
WHERE CATEGORY EQ 'Coffee'
WHERE REGION EQ 'West'
WHERE UNITS GT 1600 AND UNITS LT 1700
END
```

The output is:

PERIOD	Budget	Units	Unit Sales	Budget Dollars	Dollar Sales	EST_DOLLARS
1	1665	1678	21645	23492	0	
	1725	1669	22425	21697	0	
2	1613	1685	22582	18535	13334	
	1568	1682	23520	25230	14225	
	1847	1668	18470	25020	9607	
3	1646	1656	23044	19872	19872	
	1759	1615	17590	17765	17765	
	1498	1637	16478	21281	21281	
	1653	1694	21489	16940	16940	
4	1457	1671	21855	20052	0	
5	1662	1674	24930	18414	0	
6	1825	1695	23725	25425	-41807	
	1870	1620	24310	24300	-50319	
	1712	1640	22256	16400	-37004	
7	1727	1623	24178	17853	-25413	
	1733	1647	17330	24705	-8127	
8	1830	1652	20130	23128	6021	
	1451	1660	17412	19920	7369	
	1556	1643	18672	18073	7495	
9	1464	1663	14640	23282	11325	
	1463	1663	21945	19956	25036	
10	1464	1667	17568	25005	25005	
	1711	1623	20532	22722	22722	
	1701	1626	18711	21138	21138	
	1473	1616	14730	16160	16160	
11	1403	1601	21045	17611	0	
12	1796	1696	17960	25440	0	

Using Text Fields in DEFINE and COMPUTE

Text fields can be assigned to alphanumeric fields and receive assignment from alphanumeric fields. If an alphanumeric field is assigned the value of a text field that is too long for the alphanumeric field, the value is truncated before being assigned to the alphanumeric field.

Note: COMPUTE commands in Maintain do not support text fields.

Example: Assigning the Result of an Alphanumeric Expression to a Text Field

This example uses the COURSES data source, which contains a text field, to create an alphanumeric field named ADESC, which truncates the text field at 36 characters, and a new text field named NEWDESC, which is a text version of ADESC:

```
DEFINE FILE COURSES
ADESC/A36 = DESCRIPTION;
NEWDESC/TX36 = ADESC;
END
```

```
TABLE FILE COURSES
PRINT ADESC NEWDESC
END
```

The output is:

ADESC	NEWDESC
-----	-----
This course provides the DP professi	This course provides the DP professi
Anyone responsible for designing FOC	Anyone responsible for designing FOC
This is a course in FOCUS efficienci	This is a course in FOCUS efficienci

Creating Temporary Fields Independent of a Master File

How to:

- Display DEFINE Functions
- Define a Function
- Clear DEFINE Functions

Reference:

- DEFINE Function Limits and Restrictions

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. When calling a DEFINE function, you must first define the function.

A DEFINE function can be called in most of the same situations that are valid for Information Builders-supplied functions. Data types are defined with each argument. When substituting values for these arguments, the format must match the defined format. Alphanumeric arguments shorter than the specified format 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.

Syntax: How to Define a Function

```
DEFINE FUNCTION name (argument1/format1, ..., argumentn/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.

argument1...argumentn

Are the argument names.

format1...formatn

Are the formats of the function arguments.

If the format of an argument is alphanumeric, the argument value must also be alphanumeric. Shorter arguments are padded on the right with blanks, and longer arguments are truncated.

If the format of an argument is numeric, the argument value must also be numeric. To prevent unexpected results, you must be consistent in your use of data types.

tempvariablea...tempvariablex

Are temporary fields. Temporary fields hold intermediate values used in the function. You can define as many temporary fields as you need.

tempformata...tempformatx

Are the formats of the temporary fields.

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.

format

Is the format of the value the function returns.

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.
- ❑ A DEFINE function is cleared with the DEFINE FUNCTION CLEAR command. It is not cleared by issuing a join, or by any FOCUS command.
- ❑ When an expression tries to use a cleared function, an error appears.
- ❑ Function names are limited to eight characters.
- ❑ Argument names are limited to twelve characters. There is no limit to the number of arguments.
- ❑ DEFINE functions can call other DEFINE functions, but cannot call themselves.
- ❑ 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:

```
(FOC03665) Error loading external function '%1'
```

Example: Defining a Function

The following example creates and calls the SUBTRACT function. SUBTRACT performs a calculation with the arguments VAL1 and VAL2.

```
DEFINE FUNCTION SUBTRACT (VAL1/D8, VAL2/D8)
  SUBTRACT/D8.2 = VAL1 - VAL2;
END

TABLE FILE MOVIES
  PRINT TITLE LISTPR IN 35 WHOLESALPR AND
  COMPUTE PROFIT/D8.2 = SUBTRACT(LISTPR,WHOLESALPR);
  BY CATEGORY
  WHERE CATEGORY EQ 'MYSTERY' OR 'ACTION'
END
```

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	

Procedure: How to Display DEFINE Functions

Issue the following command from the Command Console:

```
? FUNCTION
```

Example: Displaying DEFINE Functions

Issuing the command

```
? FUNCTION
```

displays information similar to the following:

FUNCTIONS	CURRENTLY	ACTIVE	
NAME	FORMAT	PARAMETER	FORMAT
-----	-----	-----	-----
SUBTRACT	D8.2	VAL1	D8
		VAL2	D8

If you issue the ? FUNCTION command when no functions are defined, the following appears:

```
NO FUNCTIONS CURRENTLY IN EFFECT
```

Syntax: How to Clear DEFINE Functions

```
DEFINE FUNCTION {name|*} CLEAR
```

where:

name

Is the name of the function name to clear.

*

Clears all active DEFINE functions.

7 Including Totals and Subtotals

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

Topics:

- ❑ Calculating Row and Column Totals
- ❑ Including Section Totals and a Grand Total
- ❑ Including Subtotals
- ❑ Recalculating Values for Subtotal Rows
- ❑ Manipulating Summary Values With Prefix Operators
- ❑ Combinations of Summary Commands
- ❑ Producing Summary Columns for Horizontal Sort Fields
- ❑ Performing Calculations at Sort Field Breaks
- ❑ Suppressing Grand Totals
- ❑ Conditionally Displaying Summary Lines and Text

Calculating Row and Column Totals

In this section:

Producing Row Totals for Horizontal (ACROSS) Sort Field Values

How to:

Calculate Row and Column Totals

Reference:

Using ROW-TOTAL With ACROSS and Multiple Display Commands

You can produce totals for rows or columns of numbers in a report. Use:

- ❑ ROW-TOTAL to display a new column containing the sum of all numbers in each row.
- ❑ COLUMN-TOTAL to display a final row on the report, which contains the totals for each column of numbers.

You can 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 row totals using ACROSS-TOTAL.

Note that when producing totals in a report, if one field is summed, the format of the row total is the same as the format of the field. For example, if the format of the CURR_SAL field is D12.2M, the format of the row total for CURR_SAL is also D12.2M. When you are summing fields with different formats, the default format of D12.2 is used for the total.

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.

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'] [field field field]
ON TABLE ROW-TOTAL [alignment][/format] [AS 'name']
```

Example: Calculating Row and Column Totals

The following request illustrates the use of ROW-TOTAL and COLUMN-TOTAL. The column and row total labels are "TOTAL" by default. You can change them using an AS phrase.

```
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	
-----	-----	-----	-----
ASSOCIATED	\$40,680.00	\$41,620.00	\$82,300.00
BANK ASSOCIATION	\$21,780.00	\$42,962.00	\$64,742.00
BEST BANK	\$27,062.00	.	\$27,062.00
STATE	.	\$29,700.00	\$29,700.00
	\$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: 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

Reference: Using ROW-TOTAL With ACROSS and Multiple Display Commands

When a request has an ACROSS sort field, each ACROSS value displays a column for each field displayed on the report output. For example, the following request, each state has a column for units and a column for dollars:

```
TABLE FILE GGSALES
SUM UNITS AS 'U' DOLLARS AS 'D' BY CITY
ACROSS ST
IF ST EQ 'CA'
IF BUDUNITS NE MISSING
END
```

The output is:

City	State	
	CA	
	U	D
Los Angeles	298070	3772014
San Francisco	312500	3870258

When you specify a row total with ACROSS, the row total is calculated separately for each column in each ACROSS group. For example, in the following request the row total has a column for units and a column for dollars:

```
TABLE FILE GGSALES
SUM UNITS AS 'U' DOLLARS AS 'D' BY CITY
ACROSS ST
IF ST EQ 'CA'
IF BUDUNITS NE MISSING
ON TABLE ROW-TOTAL
END
```

The output is:

City	State		TOTAL	
	CA		U	D
	U	D	U	D
Los Angeles	298070	3772014	298070	3772014
San Francisco	312500	3870258	312500	3870258

When the request also has multiple display commands, each additional command adds additional columns to each ACROSS group on the report output.

The first column of the row total group is calculated by adding the first column from each display command under each ACROSS value, the second column adds the second column from each display command, and so on.

For example, the following request has a SUM command for units and dollars and another SUM command for budgeted units and budgeted dollars. The row total has a column for the sum of units and budgeted units and another column for the sum of dollars and budgeted dollars:

```
TABLE FILE GGSALES
SUM UNITS AS 'U' DOLLARS AS 'D'           BY CITY
SUM BUDUNITS AS 'BU' BUDDOLLARS AS 'BD' BY CITY
ACROSS ST
IF ST EQ 'CA'
IF BUDUNITS NE MISSING
ON TABLE ROW-TOTAL
END
```

The output is:

City	U	D	State		TOTAL	
			CA	BU	BD	BU
Los Angeles	298070	3772014	295637	3669484	593707	7441498
San Francisco	312500	3870258	314725	3916863	627225	7787121

If the different display commands do not all specify the same number of fields, some columns will not be represented in the row total. For example, in the following request, the second SUM command has a column for budgeted units but not for budgeted dollars. Therefore, the row total group has no column for dollars:

```
TABLE FILE GGSALES
SUM UNITS AS 'U' DOLLARS AS 'D' BY CITY
SUM BUDUNITS AS 'BU'           BY CITY
ACROSS ST
IF ST EQ 'CA'
IF BUDUNITS NE MISSING
ON TABLE ROW-TOTAL
END
```

The output is:

City	U	D	State		TOTAL	
			CA	BU	BU	BD
Los Angeles	298070	3772014	295637		593707	
San Francisco	312500	3870258	314725		627225	

In this case, you can use column notation to calculate the row total properly. For example, the following request calculates the row total column by adding the units, dollars, and budgeted units columns together:

```
TABLE FILE GGSALES
SUM UNITS AS 'U' DOLLARS AS 'D' BY CITY
SUM BUDUNITS AS 'BU'          BY CITY
ACROSS ST
COMPUTE TOTAL/I10 = C1 + C2 +C3; AS 'ROW-TOTAL'
IF ST EQ 'CA'
IF BUDUNITS NE MISSING
END
```

The output is:

City	U	D	State CA	BU	ROW-TOTAL
Los Angeles	298070	3772014		295637	4365721
San Francisco	312500	3870258		314725	4497483

Producing Row Totals for Horizontal (ACROSS) Sort Field Values

How to:

Produce Row Totals for Horizontal (ACROSS) Sort Field Values

Reference:

Usage Notes for ACROSS-TOTAL

You can produce row totals for horizontal (ACROSS) sort field values. Row totals for horizontal sort fields, referenced by ACROSS-TOTAL, are different from standard row totals because only horizontal sort field values, referenced by ACROSS, are included in the total. Integer, single precision floating point, double precision floating point, packed, and long packed fields can all be totaled.

Syntax: How to Produce Row Totals for Horizontal (ACROSS) Sort Field Values

```
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: Producing Row Totals for Horizontal (ACROSS) Sort Field Values

The following illustrates how to generate a row total for horizontal (ACROSS) sort field values. Notice that the summed values in the TOTAL TITLE COUNT column only reflect the values in the (RATING) PG and R columns. The values in the COPIES column are not included since they are not horizontal (ACROSS) sort field values.

```
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		TOTAL TITLE COUNT
		PG TITLE COUNT	R TITLE 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 supported.
- ❑ Attempting 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 are 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 components is five.
- ❑ ACROSS-TOTAL populates the ACROSSVALUE component in a StyleSheet. .

Including 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 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 320). You can also suppress grand totals using the NOTOTAL command. For details, see [Suppressing Grand Totals](#) on page 319.

By default, a blank line is generated before a subtotal on the report output. You can eliminate these automatic blank lines by issuing the SET DROPBLNKLINE=ON command.

Note: When the request has a PAGE-BREAK command, the GRANDTOTAL is on a page by itself.

You can use prefix operators with SUBTOTAL, SUB-TOTAL, SUMMARIZE, and RECOMPUTE. For details, see [Manipulating Summary Values With Prefix Operators](#) on page 287. In addition, you can combine different summary operations in a single request. For information, see [Combinations of Summary Commands](#) on page 306.

Example: Using Section Totals and Grand Totals

The following request illustrates how to create a subtotal every time the department value changes. The grand total is automatically produced when you use the SUBTOTAL command.

```
TABLE FILE EMPLOYEE
SUM DED_AMT BY DED_CODE BY DEPARTMENT
BY BANK_ACCT
WHERE BANK_ACCT NE 0
WHERE DED_CODE EQ 'CITY' OR 'FED'
ON DEPARTMENT SUBTOTAL
END
```

The first and last portions of the output are:

DED_CODE	DEPARTMENT	BANK_ACCT	DED_AMT
CITY	MIS	40950036	\$14.00
		122850108	\$31.75
		163800144	\$82.70
*TOTAL DEPARTMENT MIS			\$128.45
	PRODUCTION	160633	\$7.42
		136500120	\$18.25
		819000702	\$60.20
*TOTAL DEPARTMENT PRODUCTION			\$85.87
FED	MIS	40950036	\$1,190.77
		122850108	\$2,699.80
		163800144	\$7,028.30
*TOTAL DEPARTMENT MIS			\$10,918.87
	PRODUCTION	160633	\$631.12
		136500120	\$1,552.10
		819000702	\$5,120.04
*TOTAL DEPARTMENT PRODUCTION			\$7,303.26
TOTAL			\$18,436.45

Including Subtotals

How to:

Create Subtotals

Reference:

Usage Notes for 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 sort 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 319.

The subtotal is calculated every time the sort field value changes or, if WHEN criteria are applied to the sort field, every time the WHEN conditions are met.

A BY, ACROSS, or ON phrase is required to initialize the syntax.

Syntax: **How to Create Subtotals**

```
{BY|ON} fieldname {SUB-TOTAL|SUBTOTAL} [MULTILINES]  
      [field1 [AND] field2...] [AS 'text'] [WHEN expression!]
```

where:

fieldname

Must be the name of a field in a sort phrase. A BY phrase can include a summary command. 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 [Displaying Report Data](#) on page 45.

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. MULTILINES is not supported with horizontal (ACROSS) sort fields.

field1, field2, ...

Denotes a list of specific fields to subtotal. This list overrides the default, which includes all numeric display fields.

AS '*text*'

Enables you to specify a different label. For related information, see .

WHEN *expression*

Specifies the conditional display of subtotals as determined by a Boolean expression. You must end the expression with a semicolon.

Reference: Usage Notes for Subtotals

- ❑ When using a SUM or COUNT command with only one sort 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 sort phrase, SUBTOTAL is useful because there can be many values within a sort break.
- ❑ All SUB-TOTALs display up to and including the point where the sort break occurs, so 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).

- ❑ If you use a WHERE TOTAL or IF TOTAL test, the display of the sort field value for the subtotal line is suppressed unless PRINTPLUS is ON. For details about using PRINTPLUS in FOCUS, see Using PRINTPLUS in Chapter 3, Viewing and Printing Report Output.
- ❑ Subtotals display on the next line if the subtotal text does not fit on the line prior to the displayed field columns.

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	
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, the totals for DED_AMT and GROSS display only when the DEPARTMENT value changes.

The first portion of the output is:

DED_CODE	DEPARTMENT	BANK_ACCT	DED_AMT	GROSS
-----	-----	-----	-----	-----
CITY	MIS	40950036	\$14.00	\$6,099.50
		122850108	\$31.75	\$9,075.00
		163800144	\$82.70	\$22,013.75
*TOTAL DEPARTMENT MIS			\$128.45	\$37,188.25
	PRODUCTION	160633	\$7.42	\$2,475.00
		136500120	\$18.25	\$9,130.00
		819000702	\$60.20	\$17,094.00
*TOTAL DEPARTMENT PRODUCTION			\$85.87	\$28,699.00
*TOTAL DED_CODE CITY			\$214.32	\$65,887.25

The last portion of the output is:

DED_CODE	DEPARTMENT	BANK_ACCT	DED_AMT	GROSS
-----	-----	-----	-----	-----
STAT	MIS	40950036	\$196.13	\$6,099.50
		122850108	\$444.65	\$9,075.00
		163800144	\$1,157.60	\$22,013.75
*TOTAL DEPARTMENT MIS			\$1,798.38	\$37,188.25
	PRODUCTION	160633	\$103.95	\$2,475.00
		136500120	\$255.65	\$9,130.00
		819000702	\$843.32	\$17,094.00
*TOTAL DEPARTMENT PRODUCTION			\$1,202.92	\$28,699.00
*TOTAL DED_CODE STAT			\$3,001.30	\$65,887.25
TOTAL			\$41,521.18	\$461,210.75

Recalculating Values for Subtotal Rows

How to:

Subtotal Calculated Values

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 grand totals, you can include the NOTOTAL command in your request. See [Suppressing Grand Totals](#) on page 319.

Syntax: **How to Subtotal Calculated Values**

```
{BY|ON} fieldname {SUMMARIZE|RECOMPUTE} [MULTILINES]  
      [field1 [AND] field2...] [AS 'text'] [WHEN expression;]
```

where:

fieldname

Must be the name of a field in a sort phrase. A BY phrase can include a summary command. The number of fields to summarize multiplied by the number of levels of summary commands 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 [Displaying Report Data](#) on page 45.

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. MULTILINES is not supported with horizontal (ACROSS) sort fields.

You can also suppress grand totals using the NOTOTAL command, as described in [Suppressing Grand Totals](#) on page 319.

AS '*text*'

Enables you to specify a different label. For related information, see .

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. 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
```

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 portion of the output is:

PAY_DATE	DEPARTMENT	BANK_ACCT	GROSS	DED_AMT	DG_RATIO
-----	-----	-----	-----	-----	-----
82/08/31	MIS	40950036	\$1,540.00	\$725.34	.47
		122850108	\$1,815.00	\$1,261.40	.69
		163800144	\$2,255.00	\$1,668.69	.74
*TOTAL DEPARTMENT MIS			\$5,610.00	\$3,655.43	.65
	PRODUCTION	160633	\$2,475.00	\$1,427.24	.58
		136500120	\$1,342.00	\$522.28	.39
		819000702	\$2,238.50	\$1,746.03	.78
*TOTAL DEPARTMENT PRODUCTION			\$6,055.50	\$3,695.55	.61
*TOTAL PAY_DATE 82/08/31			\$11,665.50	\$7,350.98	.63

The last portion of the output is:

PAY_DATE	DEPARTMENT	BANK_ACCT	GROSS	DED_AMT	DG_RATIO
-----	-----	-----	-----	-----	-----
82/01/29	PRODUCTION	819000702	\$2,035.00	\$1,241.33	.61
*TOTAL DEPARTMENT PRODUCTION			\$2,035.00	\$1,241.33	.61
*TOTAL PAY_DATE 82/01/29			\$4,182.75	\$2,648.12	.63
81/12/31	MIS	163800144	\$2,147.75	\$1,406.79	.66
*TOTAL DEPARTMENT MIS			\$2,147.75	\$1,406.79	.66
*TOTAL PAY_DATE 81/12/31			\$2,147.75	\$1,406.79	.66
81/11/30	MIS	163800144	\$2,147.75	\$1,406.79	.66
*TOTAL DEPARTMENT MIS			\$2,147.75	\$1,406.79	.66
*TOTAL PAY_DATE 81/11/30			\$2,147.75	\$1,406.79	.66
TOTAL			\$65,887.25	\$41,521.18	.63

Tip: If you use SUB-TOTAL or SUBTOTAL rather than SUMMARIZE, the values of DG_RATIO are 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 first portion of the output is:

PAY_DATE	DEPARTMENT	BANK_ACCT	GROSS	DED_AMT	DG_RATIO
82/08/31	MIS	40950036	\$1,540.00	\$725.34	.47
		122850108	\$1,815.00	\$1,261.40	.69
		163800144	\$2,255.00	\$1,668.69	.74
*TOTAL DEPARTMENT MIS			\$5,610.00	\$3,655.43	.65
	PRODUCTION	160633	\$2,475.00	\$1,427.24	.58
		136500120	\$1,342.00	\$522.28	.39
		819000702	\$2,238.50	\$1,746.03	.78
*TOTAL DEPARTMENT PRODUCTION			\$6,055.50	\$3,695.55	.61
82/07/30	MIS	40950036	\$1,540.00	\$725.34	.47
		122850108	\$1,815.00	\$1,261.40	.69

The last portion of the output is:

PAY_DATE	DEPARTMENT	BANK_ACCT	GROSS	DED_AMT	DG_RATIO
82/01/29	MIS	163800144	\$2,147.75	\$1,406.79	.66
*TOTAL DEPARTMENT MIS			\$2,147.75	\$1,406.79	.66
	PRODUCTION	819000702	\$2,035.00	\$1,241.33	.61
*TOTAL DEPARTMENT PRODUCTION			\$2,035.00	\$1,241.33	.61
81/12/31	MIS	163800144	\$2,147.75	\$1,406.79	.66
*TOTAL DEPARTMENT MIS			\$2,147.75	\$1,406.79	.66
81/11/30	MIS	163800144	\$2,147.75	\$1,406.79	.66
*TOTAL DEPARTMENT MIS			\$2,147.75	\$1,406.79	.66
TOTAL			\$65,887.25	\$41,521.18	.63

Manipulating Summary Values With Prefix Operators

In this section:

Controlling Summary Line Processing

Using Prefix Operators With Calculated Values

Using Multiple SUB-TOTAL or SUMMARIZE Commands With Prefix Operators

How to:

Use Prefix Operators With Summary Values

Reference:

Usage Notes for Summary Prefix Operators

You can use the SUBTOTAL, SUB-TOTAL, RECOMPUTE, and SUMMARIZE commands at the ON TABLE level to specify the type of summary operation to use to produce the grand total line on the report.

In addition, prefix operators can be used with the summary options SUBTOTAL, SUB-TOTAL, RECOMPUTE, and SUMMARIZE at both the sort break and grand total levels.

Prefix operations on summary lines are performed on the retrieved, selected, and summed values that become the detail lines in the report. Unlike field-based prefix operations, they are not performed on each incoming record.

Each type of summary has its own purpose, and handles the prefix operators appropriately for the type of summary information to be displayed. For example, using AVE. at a sort field break produces the average within the sort group.

Alphanumeric fields can also be displayed on summary lines. In order to do this, you must either explicitly list the alphanumeric field name on the summary command, or use the asterisk (*) wildcard to display all fields.

Different operations from two ON phrases for the same sort break display on the same summary line, and allow a mixture of operations on summary lines. The grand total line populates all fields populated by any summary command, even fields that are not specified in the grand total command.

If the same field is referenced in more than one ON phrase for the same sort break, the last function specified is applied.

The following prefix operators are supported for numeric fields:

- ❑ ASQ.
- ❑ AVE.

- ❑ CNT.
- ❑ FST.
- ❑ LST.
- ❑ MAX.
- ❑ MIN.
- ❑ SUM.

The following prefix operators are supported for alphanumeric fields:

- ❑ FST.
- ❑ LST.
- ❑ MAX.
- ❑ MIN.
- ❑ SUM. (means LST. if SUMPREFIX=LST or FST. if SUMPREFIX=FST)

Syntax: **How to Use Prefix Operators With Summary Values**

```
{BY|ON} breakfield [AS 'text1'] sumoption [MULTILINES]
      [pref. ] [*|[[field1 [[pref2. ] field2 ...]]]
      [AS 'text2'] [WHEN expression;]
```

To replace the default grand total, use the following syntax

```
ON TABLE sumoption [pref. ][field1 [[pref2. ]field2 ...]] [AS 'text2']
```

where:

breakfield

Is the sort field whose change in value triggers the summary operation. A BY phrase can include a summary command. When the value of the sort field changes, it triggers the summary operation.

sumoption

Can be one of the following: SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE.

'text1'

Is the column heading to use for the break field on the report output.

MULTILINES

Suppresses the printing of a summary line for every sort break that has only one detail line. Note that MULTILINES suppresses the summary line even if a prefix operator is used to specify a different operation for the summary line. MULTI-LINES is a synonym for MULTILINES. MULTILINES is not supported with horizontal (ACROSS) sort fields.

pref.

Is a prefix operator. When specified without a field list, the prefix operator is applied to every numeric column in the report output and every numeric column is populated with values on the summary row.

*

Includes all display fields on the summary line. If a prefix operator is specified, it is applied to all fields. If the prefix operator is not supported with alphanumeric fields, alphanumeric fields are not included on the summary line.

[field1 [field2 ... fieldn]]

Produces the type of summary specified by *sumoption* for the listed fields. If no field names are listed, the summary is produced for every numeric column in the report output.

pref. field1 [field2 ... fieldn] [pref2. fieldm ...]

The first prefix operator is applied to field1 through fieldn. The second prefix operator is applied to fieldm. Only the fields specified are populated with values on the summary row. Each prefix operator must be separated by a blank space from the following field name. For example:



 BY STORE_CODE SUMMARIZE AVE. UNIT_SOLD RETURNS CNT. DAMAGED

'text2'

Is the text that prints on the left of the summary row.

expression

Is an expression that determines whether the summary operation is performed at each break.

Reference: Usage Notes for Summary Prefix Operators

- COLUMN-TOTAL does not support prefix operators.
- Prefix operators PCT., RPCT., AND TOT. are not supported.
- Double prefix operators (such as PCT.CNT.) are not supported.

- ❑ When an ACROSS field is used in the request, the same field name displays over multiple columns (ACROSS groups) in the report output. A prefix operator applied to such a field on a summary line is applied to all of those columns.
- ❑ The SUM. prefix operator produces the same summary values as a summary phrase with no prefix operator.
- ❑ SUMMARIZE and RECOMPUTE apply the calculations defined in the associated COMPUTE command to the summary values. Therefore, in order to perform the necessary calculations, the SUMMARIZE or RECOMPUTE command must calculate all of the fields referenced in the COMPUTE command.
- ❑ If the same field is referenced by more than one summary operation with different prefix operators at each level, the default grand total (one produced without an ON TABLE summaryoption command) applies the operation specified by the first operator used in the report request (the left-most sort field in the output).

Example: Using Prefix Operators With SUBTOTAL

The following example uses prefix operators to calculate the:

- ❑ Average list price by rating.
- ❑ Sum copies by category within the rating field.

Notice that the subtotal row for each rating contains a value only in the LISTPR column, and the subtotal row for each category contains a value only in the COPIES column. The grand total line contains values only for the columns that were subtotaled. Note the blank space between each prefix operator and the field name that follows it:

```
TABLE FILE MOVIES
PRINT COPIES LISTPR WHOLESALPR TITLE/A23
  BY RATING
  BY CATEGORY
  WHERE CATEGORY EQ 'CHILDREN' OR 'CLASSIC'
  WHERE RATING EQ 'G' OR 'NR'
  ON RATING SUBTOTAL AVE. LISTPR AS '*Ave: '
  ON CATEGORY SUBTOTAL SUM. COPIES AS '*Sum: '
END
```

The output is:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALEPR	TITLE
-----	-----	-----	-----	-----	-----
G	CHILDREN	2	44.95	29.99	SHAGGY DOG, THE
		2	29.95	12.50	ALICE IN WONDERLAND
		3	26.99	12.00	BAMBI
*Sum:	CHILDREN	7			
	CLASSIC	3	89.95	40.99	GONE WITH THE WIND
*Sum:	CLASSIC	3			
*Ave:	G		47.96		
NR	CHILDREN	1	19.95	10.00	SMURFS, THE
		1	19.95	9.75	SCOOBY-DOO-A DOG IN THE
		1	14.95	7.65	SESAME STREET-BEDTIME S
		1	14.98	7.99	ROMPER ROOM-ASK MISS MO
		1	29.95	15.99	SLEEPING BEAUTY
*Sum:	CHILDREN	5			
	CLASSIC	1	24.98	14.99	EAST OF EDEN
		3	39.99	20.00	CITIZEN KANE
		1	29.95	15.99	CYRANO DE BERGERAC
		1	19.99	10.95	MARTY
		2	19.99	10.95	MALTESE FALCON, THE
		2	19.95	9.99	ON THE WATERFRONT
		2	89.99	40.99	MUTINY ON THE BOUNTY
		2	19.99	10.95	PHILADELPHIA STORY, THE
		2	19.98	10.99	CAT ON A HOT TIN ROOF
		2	29.95	15.00	CASABLANCA
*Sum:	CLASSIC	18			
*Ave:	NR		27.64		
TOTAL		33	31.91		

Example: Using SUBTOTAL at the Sort Break and Grand Total Levels

The following example adds the ON TABLE SUBTOTAL command to the request in *Using Prefix Operators With SUBTOTAL* on page 290 at the sort break level to calculate the minimum number of copies and maximum list price on the grand total line for the entire report:

```
TABLE FILE MOVIES
PRINT COPIES LISTPR WHOLESALPR TITLE/A23
  BY RATING
  BY CATEGORY
  WHERE CATEGORY EQ 'CHILDREN' OR 'CLASSIC'
  WHERE RATING EQ 'G' OR 'NR'
  ON RATING SUBTOTAL AVE. LISTPR AS '*Ave: '
  ON CATEGORY SUBTOTAL SUM. COPIES AS '*Sum: '
  ON TABLE SUBTOTAL MIN. COPIES MAX. LISTPR
END
```

The output is exactly the same as in the previous request, except for the grand total line:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR	TITLE
G	CHILDREN	2	44.95	29.99	SHAGGY DOG, THE
		2	29.95	12.50	ALICE IN WONDERLAND
		3	26.99	12.00	BAMBI
*Sum:	CHILDREN	7			
	CLASSIC	3	89.95	40.99	GONE WITH THE WIND
*Sum:	CLASSIC	3			
*Ave:	G		47.96		
NR	CHILDREN	1	19.95	10.00	SMURFS, THE
		1	19.95	9.75	SCOOBY-DOO-A DOG IN THE
		1	14.95	7.65	SESAME STREET-BEDTIME S
		1	14.98	7.99	ROMPER ROOM-ASK MISS MO
		1	29.95	15.99	SLEEPING BEAUTY
*Sum:	CHILDREN	5			
	CLASSIC	1	24.98	14.99	EAST OF EDEN
		3	39.99	20.00	CITIZEN KANE
		1	29.95	15.99	CYRANO DE BERGERAC
		1	19.99	10.95	MARTY
		2	19.99	10.95	MALTESE FALCON, THE
		2	19.95	9.99	ON THE WATERFRONT
		2	89.99	40.99	MUTINY ON THE BOUNTY
		2	19.99	10.95	PHILADELPHIA STORY, THE
		2	19.98	10.99	CAT ON A HOT TIN ROOF
		2	29.95	15.00	CASABLANCA
*Sum:	CLASSIC	18			
*Ave:	NR		27.64		
TOTAL		1	89.99		

Example: Displaying an Alphanumeric Field on a Summary Line

The following request displays the sum of the list price field and the minimum value of the director field by rating:

```
TABLE FILE MOVIES
PRINT COPIES LISTPR WHOLESALPR DIRECTOR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN' OR 'CLASSIC'
WHERE RATING EQ 'G' OR 'NR'
WHERE DIRECTOR NE ' '
ON RATING SUBTOTAL SUM. LISTPR MIN. DIRECTOR AS '*A/N:'
END
```

The output is:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR	DIRECTOR
-----	-----	-----	-----	-----	-----
G	CHILDREN	2	44.95	29.99	BARTON C.
		2	29.95	12.50	GEROMINI
		3	26.99	12.00	DISNEY W.
	CLASSIC	3	89.95	40.99	FLEMING V
*A/N: G			191.84		BARTON C.
NR	CHILDREN	1	29.95	15.99	DISNEY W.
	CLASSIC	1	24.98	14.99	KAZAN E.
		3	39.99	20.00	WELLES O.
		1	29.95	15.99	GORDON M.
		1	19.99	10.95	MANN D.
		2	19.99	10.95	HUSTON J.
		2	19.95	9.99	KAZAN E.
		2	89.99	40.99	MILESTONE L.
		2	19.99	10.95	CUKOR G.
		2	19.98	10.99	BROOKS R.
		2	29.95	15.00	CURTIZ M.
*A/N: NR			344.71		BROOKS R.
TOTAL			536.55		BARTON C.

Example: Displaying All Fields on a Summary Line

The following request displays the sum of every display field on the subtotal line. The director field is alphanumeric, so the last value displays:

```
TABLE FILE MOVIES
PRINT COPIES LISTPR WHOLESALPR DIRECTOR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN' OR 'CLASSIC'
WHERE RATING EQ 'G' OR 'NR'
WHERE DIRECTOR NE ' '
ON RATING SUBTOTAL SUM. * AS '*All: '
END
```

The output is:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR	DIRECTOR
-----	-----	-----	-----	-----	-----
G	CHILDREN	2	44.95	29.99	BARTON C.
		2	29.95	12.50	GEROMINI
		3	26.99	12.00	DISNEY W.
	CLASSIC	3	89.95	40.99	FLEMING V
*All:	G	10	191.84	95.48	FLEMING V
NR	CHILDREN	1	29.95	15.99	DISNEY W.
	CLASSIC	1	24.98	14.99	KAZAN E.
		3	39.99	20.00	WELLES O.
		1	29.95	15.99	GORDON M.
		1	19.99	10.95	MANN D.
		2	19.99	10.95	HUSTON J.
		2	19.95	9.99	KAZAN E.
		2	89.99	40.99	MILESTONE L.
		2	19.99	10.95	CUKOR G.
		2	19.98	10.99	BROOKS R.
		2	29.95	15.00	CURTIZ M.
*All:	NR	19	344.71	176.79	CURTIZ M.
TOTAL		29	536.55	272.27	CURTIZ M.

Controlling Summary Line Processing

How to:

Control Summary Line Processing

Reference:

Usage Notes for SET SUMMARYLINES

When processing summary lines, you can control whether prefix operator processing is used and whether SUBTOTAL and RECOMPUTE commands are propagated to the grand total row of a report.

Summary line processing with prefix operators differs from processing without prefix operators in both the types of operations supported and the fields affected.

By default, you cannot mix these two styles of processing in a request, and the syntax used in the request (prefix operators or no prefix operators on summary lines) determines which type of processing is used.

One function of the SUMMARYLINES setting is to allow you to combine fields with and without prefix operators on summary lines in one request. In this case, prefix operator processing is used for all summary lines. Those fields without prefix operators are processed as though they were specified with the operator SUM. The new processing is required to display alphanumeric fields on summary lines.

Processing of reports that use prefix operators on summary lines differs from processing without prefix operators. In some cases, a different style of report output results from each type of request.

If one summary command specifies one field name and another summary command specifies a second field name:

- ❑ In a report without summary prefix operators, both columns are populated on both summary lines.
- ❑ In a report with summary prefix operators, only the specified column is populated on each summary line.

If a prefix operator is used in any summary command, prefix operator processing is required for the request. In most requests it is clear which type of processing to use, even if prefix operators are specified in some summary commands but not in others.

However, if the first time a summary prefix operator is encountered occurs after a field name has been specified in a summary command without an accompanying prefix operator, neither type of processing can be implemented. In this case, by default, processing stops and the following error message is generated:

```
(FOC36376) CANNOT COMBINE SUBTOTAL/RECOMPUTE STYLES WHEN SUMMARYLINES=OLD
```

For example:

```
ON RATING SUBTOTAL COPIES AVE. LISTPR
```

or

```
ON RATING SUBTOTAL LISTPR  
ON CATEGORY SUBTOTAL AVE. COPIES
```

You can eliminate this problem by issuing the SET SUMMARYLINES=NEW command to specify that prefix operator processing should be used. The SUM. operator is then applied to any field that does not have a prefix operator.

The new processing is required to display alphanumeric fields on summary lines.

The second function of the SET SUMMARYLINES command is to make the processing of SUBTOTAL, SUB-TOTAL, SUMMARIZE, and RECOMPUTE on the grand total line consistent with how they work for sort field breaks. The setting that invokes this type of processing is SET SUMMARYLINES=EXPLICIT.

When SUBTOTAL and RECOMPUTE are used at a sort break level, they do not propagate to other sort breaks. SUB-TOTAL and SUMMARIZE propagate to all higher level sort breaks.

The grand total can be considered the highest level sort field in a request. However, by default, all of the summary options, not just SUB-TOTAL and SUMMARIZE, propagate to the grand total level.

The SET SUMMARYLINES=EXPLICIT command prevents the propagation of SUBTOTAL and RECOMPUTE to the grand total. In addition, if all summary commands in the request specify field lists, only the specified fields are aggregated and displayed on the grand total line.

When SUBTOTAL and RECOMPUTE are the only summary commands used in the request, a grand total line is produced only if it is explicitly specified in the request using the ON TABLE SUBTOTAL/SUB-TOTAL/RECOMPUTE/SUMMARIZE phrase. If the ON TABLE phrase specifies a field list, only those fields are aggregated and displayed.

Note that you can always suppress the grand total line using the ON TABLE NOTOTAL command in the request.

Syntax: How to Control Summary Line Processing

```
SET SUMMARYLINES = {OLD|NEW|EXPLICIT}
```

where:

OLD

Does not allow summary fields with and without prefix operators and propagates all summary operations to the grand total line. Fields specified on summary lines must all have prefix operators applied or must all exclude them. This syntax determines which type of processing is applied. OLD is the default value.

Note that a prefix operator preceding a list of field names is applied to each of those report columns (and, therefore, is not considered mixing). You can specify the SUM. operator for fields for which you want a standard subtotal. This produces the same value that would have been generated without prefix operators. Alphanumeric fields are not included on summary lines.

NEW

Propagates all summary operations to the grand total line. Uses prefix operator processing for all summary commands (all summary fields without prefix operators are processed as though they had a SUM. operator). Fields listed in a summary command are populated only on summary lines created by that summary command and on summary lines created by propagation of that summary command. Supports display of alphanumeric fields on summary lines.

EXPLICIT

Does not propagate SUBTOTAL and RECOMPUTE to the grand total line. Uses prefix operator processing for all summary commands (all summary fields without prefix operators are processed as though they had a SUM. operator). Fields listed in a summary command are populated only on summary lines created by that summary command and on summary lines created by propagation of that summary command. Supports display of alphanumeric fields on summary lines.

Note: This command is not supported in a request using the ON TABLE SET syntax.

Reference: Usage Notes for SET SUMMARYLINES

- ❑ SET SUMMARYLINES is not supported within a TABLE request (ON TABLE).
- ❑ If COLUMN-TOTAL is specified in the request, all numeric fields are totaled on the grand total line unless the COLUMN-TOTAL phrase lists specific fields. If the COLUMN-TOTAL phrase lists specific fields, those fields and any fields propagated by SUB-TOTAL or SUMMARIZE commands are totaled.

- ❑ Even if prefix operators are not used on summary lines, report output generated by the two settings for the same request may be slightly different. With SUMMARYLINES NEW, a summary command with a list of field names populates only those columns on the associated summary line, while SUMMARYLINES OLD populates every column specified in any summary command.

For example:

```
TABLE FILE MOVIES
PRINT COPIES LISTPR WHOLESALPR
  BY RATING
  BY CATEGORY
  WHERE CATEGORY EQ 'CHILDREN'
  WHERE RATING EQ 'G'
  ON RATING SUBTOTAL LISTPR AS '*LIST'
  ON CATEGORY SUBTOTAL COPIES AS '*COPY'
END
```

The output when SUMMARYLINES=OLD has subtotals for both COPIES and LISTPR on both sort breaks. WHOLESALPR is not referenced in either SUBTOTAL command and, therefore, is not on any summary line:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR
G	CHILDREN	2	44.95	29.99
		2	29.95	12.50
		3	26.99	12.00
*COPY CHILDREN		7	101.89	
*LIST G		7	101.89	
TOTAL		7	101.89	

The output when SUMMARYLINES=NEW has subtotals for COPIES on the CATEGORY sort break and for LISTPR on the RATING sort break. Both columns are populated on the grand total line. WHOLESALPR is not referenced in either SUBTOTAL command and, therefore, is not on any summary line:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR
G	CHILDREN	2	44.95	29.99
		2	29.95	12.50
		3	26.99	12.00
*COPY CHILDREN		7		
*LIST G			101.89	
TOTAL		7	101.89	

Example: Using SET SUMMARYLINES With SUBTOTAL

The following request using the MOVIES data source has a sort break for CATEGORY that subtotals the COPIES field and a sort break for RATING that subtotals the LISTPR field:

```
SET SUMMARYLINES=OLD
TABLE FILE MOVIES
SUM COPIES LISTPR WHOLESALPR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN'
WHERE RATING EQ 'G'
ON RATING SUBTOTAL COPIES
ON CATEGORY SUBTOTAL LISTPR
END
```

Running the request with SUMMARYLINES=OLD subtotals both COPIES and LISTPR at every sort break and propagates them to the grand total line:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR
G	CHILDREN	7	101.89	54.49
*TOTAL CHILDREN		7	101.89	
*TOTAL G		7	101.89	
TOTAL		7	101.89	

Running the request with SUMMARYLINES=NEW subtotals COPIES only for the RATING sort break and subtotals LISTPR only for the CATEGORY sort break but propagates both to the grand total line:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR
G	CHILDREN	7	101.89	54.49
*TOTAL CHILDREN			101.89	
*TOTAL G		7		
TOTAL		7	101.89	

Running the request with SUMMARYLINES=EXPLICIT subtotals COPIES only for the RATING sort break and subtotals LISTPR only for the CATEGORY sort break. It does not produce a grand total line:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR
G	CHILDREN	7	101.89	54.49
*TOTAL CHILDREN			101.89	
*TOTAL G		7		

Adding the phrase ON TABLE SUBTOTAL WHOLESALPR with SUMMARYLINES=EXPLICIT produces a grand total line with the WHOLESALPR field subtotaled:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR
G	CHILDREN	7	101.89	54.49
*TOTAL CHILDREN			101.89	
*TOTAL G		7		
TOTAL				54.49

Example: Using COLUMN-TOTAL With SET SUMMARYLINES=EXPLICIT

The following request using the MOVIES data source has a sort break for CATEGORY for which subtotals the COPIES field and a sort break for RATING that subtotals the LISTPR field. It also has an ON TABLE COLUMN-TOTAL phrase:

```
SET SUMMARYLINES=EXPLICIT
TABLE FILE MOVIES
SUM COPIES LISTPR WHOLESALPR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN'
WHERE RATING EQ 'G'
ON RATING SUBTOTAL COPIES
ON CATEGORY SUBTOTAL LISTPR
ON TABLE COLUMN-TOTAL
END
```

The grand total line displays a column total for all numeric columns because of the ON TABLE COLUMN-TOTAL phrase:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR
G	CHILDREN	7	101.89	54.49
*TOTAL CHILDREN			101.89	
*TOTAL G		7		
TOTAL		7	101.89	54.49

The following request has an ON TABLE SUBTOTAL WHOLESALPR command. It also has an ON TABLE COLUMN-TOTAL phrase:

```
SET SUMMARYLINES=EXPLICIT
TABLE FILE MOVIES
SUM COPIES LISTPR WHOLESALPR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN'
WHERE RATING EQ 'G'
ON RATING SUBTOTAL COPIES
ON CATEGORY SUBTOTAL LISTPR
ON TABLE SUBTOTAL WHOLESALPR
ON TABLE COLUMN-TOTAL
END
```

The grand total line displays a column total only for the WHOLESALPR column because of the ON TABLE SUBTOTAL command:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALPR
-----	-----	-----	-----	-----
G	CHILDREN	7	101.89	54.49
*TOTAL CHILDREN			101.89	
*TOTAL G		7		
TOTAL				54.49

Using SUB-TOTAL instead of SUBTOTAL causes COPIES and LISTPR to be aggregated on the grand total line. WHOLESALPR is totaled because it is listed in the COLUMN-TOTAL phrase. The subtotal for LISTPR propagates to the RATING sort break as well as to the grand total:

```
SET SUMMARYLINES=EXPLICIT
TABLE FILE MOVIES
SUM COPIES LISTPR WHOLESALPR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN'
WHERE RATING EQ 'G'
ON RATING SUB-TOTAL COPIES
ON CATEGORY SUB-TOTAL LISTPR
ON TABLE COLUMN-TOTAL WHOLESALPR
END
```

The output is:

RATING	CATEGORY	COPIES	LISTPR	WHOLESALEPR
G	CHILDREN	7	101.89	54.49
*TOTAL	CHILDREN		101.89	
*TOTAL	G	7	101.89	
TOTAL		7	101.89	54.49

Using Prefix Operators With Calculated Values

If a request includes the RECOMPUTE or SUMMARIZE command, the expression specified in the associated COMPUTE command is applied using values from the summary line. The columns used to recompute the expression can have prefix operators. The recomputed column, regardless of the prefix operator specified for it, applies these input values to the expression specified in the COMPUTE command. Therefore, any supported prefix operator can be specified for the recomputed report column without affecting the calculated value.

With prefix operator processing, all fields used in the COMPUTE command must be displayed by the RECOMPUTE or SUMMARIZE command in order to be populated. If any field used in the expression is not populated, the calculated value returned for the expression is unpredictable.

Example: Using Prefix Operators With RECOMPUTE

The first request creates a calculated field named , which is the difference between DOLLARS and BUDDOLLARS. This value is then recomputed for each region, without using prefix operators.

```
TABLE FILE GGSales
SUM UNITS DOLLARS BUDDOLLARS
AND COMPUTE DIFF/I10 = DOLLARS-BUDDOLLARS;
  BY REGION
  BY CATEGORY
  WHERE CATEGORY EQ 'Food' OR 'Coffee'
  WHERE REGION EQ 'West' OR 'Midwest'
  ON REGION RECOMPUTE
END
```

The recomputed value is the difference between the totals for DOLLARS and BUDDOLLARS.

Region	Category	Unit Sales	Dollar Sales	Budget Dollars	DIFF
Midwest	Coffee	332777	4178513	4086032	92481
	Food	341414	4338271	4220721	117550
*TOTAL Midwest		674191	8516784	8306753	210031
West	Coffee	356763	4473517	4523963	-50446
	Food	340234	4202337	4183244	19093
*TOTAL West		696997	8675854	8707207	-31353
TOTAL		1371188	17192638	17013960	178678

The following request uses prefix operators in the RECOMPUTE command to calculate the maximum DOLLARS and the minimum BUDDOLLARS and then recompute DIFF. No matter which prefix operator we specify for DIFF, it is calculated as the difference between the values in the DOLLARS and BUDDOLLARS columns. If any of the fields used in the calculation (DOLLARS, BUDDOLLARS, and DIFF) do not display on the summary row, the calculation cannot be performed.

```
TABLE FILE GGSales
SUM UNITS DOLLARS BUDDOLLARS
AND COMPUTE DIFF/I10 = DOLLARS-BUDDOLLARS;
  BY REGION
  BY CATEGORY
  WHERE CATEGORY EQ 'Food' OR 'Coffee'
  WHERE REGION EQ 'West' OR 'Midwest'
  ON REGION RECOMPUTE MAX. DOLLARS MIN. BUDDOLLARS AVE. DIFF
END
```

The output is:

Region	Category	Unit Sales	Dollar Sales	Budget Dollars	DIFF
Midwest	Coffee	332777	4178513	4086032	92481
	Food	341414	4338271	4220721	117550
*TOTAL Midwest			4338271	4086032	252239
West	Coffee	356763	4473517	4523963	-50446
	Food	340234	4202337	4183244	19093
*TOTAL West			4473517	4183244	290273

Example: Using RECOMPUTE at the Sort Break and Grand Total Levels

The following example adds the ON TABLE RECOMPUTE command to the request in *Using Prefix Operators With RECOMPUTE* on page 302 to calculate the average values for each column. Notice that the value of DIFF is calculated as the difference between the values in the Dollar Sales and the Budget Dollars columns on the grand total line:

```
TABLE FILE GGSALES
SUM UNITS DOLLARS BUDDOLLARS
AND COMPUTE DIFF/I10 = DOLLARS-BUDDOLLARS;
  BY REGION
  BY CATEGORY
  WHERE CATEGORY EQ 'Food' OR 'Coffee'
  WHERE REGION EQ 'West' OR 'Midwest'
  ON REGION RECOMPUTE MAX. DOLLARS MIN. BUDDOLLARS DIFF
ON TABLE RECOMPUTE AVE.
END
```

The output is:

Region	Category	Unit Sales	Dollar Sales	Budget Dollars	DIFF
Midwest	Coffee	332777	4178513	4086032	92481
	Food	341414	4338271	4220721	117550
*TOTAL Midwest			4338271	4086032	252239
West	Coffee	356763	4473527	4523963	-50436
	Food	340234	4202338	4183244	19094
*TOTAL West			4473527	4183244	290283
TOTAL		342797	4298162	4253490	44672

Using Multiple SUB-TOTAL or SUMMARIZE Commands With Prefix Operators

SUB-TOTAL and SUMMARIZE propagate their operations to all higher-level sort fields. If a request uses SUB-TOTAL or SUMMARIZE at multiple sort levels, more than one prefix operator may apply to the same field.

When a SUB-TOTAL or SUMMARIZE command on a lower-level sort field propagates up to the higher levels, it applies its prefix operators only to those fields that did not already have different prefix operators specified at the higher level. For any field that had a prefix operator specified at a higher level, the original prefix operator is applied at the level at which it was first specified and to the grand total line, unless a different operator is specified for the grand total line.

Example: Using Multiple SUB-TOTAL Commands With Prefix Operators

The following illustrates prefix operators work in a request that has multiple SUB-TOTAL commands, each with a different prefix operator.

```

DEFINE FILE GGSALES
YEAR/YY = DATE;
END

TABLE FILE GGSALES
SUM  UNITS DOLLARS/D10.2 BUDDOLLARS
  BY YEAR
  BY ST
  BY REGION
  BY CATEGORY
WHERE REGION EQ 'West' OR 'Midwest'
WHERE ST     EQ 'CA' OR 'IL'
WHERE YEAR  EQ '1996' OR '1997'
  ON YEAR SUB-TOTAL CNT. UNITS AS '*CNT. UNITS:'
  ON ST   SUB-TOTAL AVE. DOLLARS AS '*AVE. $:'
  ON REGION SUB-TOTAL MIN. AS '*MIN.:'
END

```

In the following report output, some of the values have been manually italicized or bolded for clarity:

- ❑ Outlined rows are the rows generated by the SUB-TOTAL commands.
- ❑ Subtotal values in the normal typeface are the count of unit sales generated by the command ON YEAR SUB-TOTAL CNT. UNITS. This is the topmost summary command, and therefore does not propagate to any other summary lines.
- ❑ Subtotal values in italic are average dollar sales generated by the command ON ST SUB-TOTAL AVE. DOLLARS. This is the second summary command, and therefore propagates to the DOLLARS column of summary lines for the YEAR sort field.
- ❑ Subtotal values in boldface are minimums within their sort groups generated by the command ON REGION SUB-TOTAL MIN. This is the last summary command, and therefore propagates to all other summary lines, but only calculates minimum values for those columns not already populated with a count or an average.

<u>YEAR</u>	<u>State</u>	<u>Region</u>	<u>Category</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>	<u>Budget</u>	<u>Dollars</u>
1996	CA	West	Coffee	117539	1,484,873.00		1453548
			Food	116389	1,443,083.00		1414902
			Gifts	74948	947,783.00		946382
				74948	947,783.00		946382
			*MIN.: West				
			*AVE. \$: CA	74948	1,291,913.00		946382
	IL	Midwest	Coffee	52348	683,559.00		628333
			Food	58777	768,715.00		742943
			Gifts	38405	481,515.00		487090
				38405	481,515.00		487090
			*MIN.: Midwest				
			*AVE. \$: IL	38405	644,596.33		487090
			*CNT. UNITS: 1996	6	968,254.67		487090
1997	CA	West	Coffee	118044	1,453,013.00		1507092
			Food	106322	1,325,429.00		1302582
			Gifts	77328	988,080.00		961841
				77328	988,080.00		961841
			*MIN.: West				
			*AVE. \$: CA	77328	1,255,507.33		961841
	IL	Midwest	Coffee	57233	715,220.00		737931
			Food	59293	754,132.00		737912
			Gifts	41527	521,260.00		532647
				41527	521,260.00		532647
			*MIN.: Midwest				
			*AVE. \$: IL	41527	663,537.33		532647
			*CNT. UNITS: 1997	6	959,522.33		532647
			TOTAL	12	963,888.50		487090

Combinations of Summary Commands

Reference:

Usage Notes for Combinations of Summary Commands

You can specify a different summary operation for each sort break (BY or ACROSS field).

If you have multiple summary commands for the same sort field, the following message displays and the last summary command specified in the request is used:

```
(FOC36359) MORE THAN 1 SUBTOTAL/SUB-TOTAL/RECOMPUTE/SUMMARIZE
```

There is more than one SUBTOTAL/SUB-TOTAL/RECOMPUTE/SUMMARIZE on the same key field which is not allowed. The last one specified will override the rest.

SUMMARIZE and SUB-TOTAL, which propagate their summary operations to higher level sort breaks, skip those fields at higher level sort breaks that have their own summary commands. The propagation of summary operations depends on whether prefix operator processing is used for summary lines. If prefix operators are:

- ❑ Not used on summary lines or if you issue the SET SUMMARYLINES=OLD command, prefix operator processing is not used for the request. In this case, if any summary command specifies a field list, only the fields specified on the summary line field lists are populated on the report.
- ❑ Used on summary lines or if you issue the SET SUMMARYLINES=NEW command, prefix operator processing is used for the request. In this case, SUB-TOTAL and SUMMARIZE propagate to:
 - ❑ All fields at higher level sort breaks that do not have their own summary command.
 - ❑ Fields not specified in the field list at higher level sort breaks that do have their own summary commands (columns that would have been empty). Note that this is the only technique that allows different fields at the same sort break to have different summary options.

Prefix operators on summary lines result in the same values whether the command is RECOMPUTE/SUMMARIZE or SUBTOTAL/SUB-TOTAL. For a computed field, the prefix operator is not applied, the value is recalculated using the expression in the COMPUTE command and the values from the summary line.

When you use different summary commands for different sort fields, the default grand total row inherits the summary command associated with the first sort field in the request. You can change the operation performed at the grand total level by using the ON TABLE phrase to specify a specific summary command.

Note: The grand total is considered the highest sort level. Therefore, although you can use the SUMMARIZE or SUB-TOTAL command at the grand total level, these commands apply only to the grand total and are not propagated to any other line on the report. On the grand total level SUMMARIZE operates as a RECOMPUTE command, and SUB-TOTAL operates as a SUBTOTAL command.

Example: Using SUBTOTAL and RECOMPUTE in a Request

In the following request, the first sort field specified is COPIES, which is associated with the RECOMPUTE command. Therefore, on the grand total line, the value of RATIO is correctly recomputed and the values of LISTPR and WHOLESALPR are summed (because this is the default operation when the field is not calculated by a COMPUTE command).

```
TABLE FILE MOVIES
PRINT DIRECTOR LISTPR WHOLESALPR
COMPUTE RATIO = LISTPR/WHOLESALPR;
BY COPIES
BY RATING
WHERE COPIES LT 3
WHERE DIRECTOR EQ 'DISNEY W.' OR 'HITCHCOCK A.'
ON COPIES RECOMPUTE AS '*REC: '
ON RATING SUBTOTAL AS '*SUB: '
END
```

The output is:

COPIES	RATING	DIRECTOR	LISTPR	WHOLESALPR	RATIO
1	NR	DISNEY W.	29.95	15.99	1.87
*SUB:	NR		29.95	15.99	1.87
*REC:	1		29.95	15.99	1.87
2	NR	HITCHCOCK A.	19.98	9.00	2.22
*SUB:	NR		19.98	9.00	2.22
	PG	HITCHCOCK A.	19.98	9.00	2.22
		HITCHCOCK A.	19.98	9.00	2.22
*SUB:	PG		39.96	18.00	4.44
2	PG13	HITCHCOCK A.	19.98	9.00	2.22
*SUB:	PG13		19.98	9.00	2.22
	R	HITCHCOCK A.	19.98	9.00	2.22
*SUB:	R		19.98	9.00	2.22
*REC:	2		99.90	45.00	2.22
TOTAL			129.85	60.99	2.13

If you reverse the BY fields, the grand total line sums the RATIO values as well as the LISTPR and WHOLESALPR values because the SUBTOTAL command controls the grand total line:

TOTAL			129.85	60.99	12.97
-------	--	--	--------	-------	-------

You can change the operation performed at the grand total level by adding the following command to the request:

```
ON TABLE RECOMPUTE
```

The grand total line then displays the recomputed values:

TOTAL	129.85	60.99	2.13
-------	--------	-------	------

Example: Using SUB-TOTAL With Multiple Summary Commands

In the following request, the SUB-TOTAL command propagates its operation to the DIRECTOR sort field (see the total line for HITCHCOCK, on which the RATIO values are subtotaled, not recomputed).

SUB-TOTAL is not propagated to the RATING sort field which has its own RECOMPUTE command, and for this sort field the RATIO value is recomputed. The grand total line is recomputed because RECOMPUTE is performed on a higher level sort field than SUB-TOTAL.

```
TABLE FILE MOVIES
PRINT LISTPR WHOLESALPR
COMPUTE RATIO = LISTPR/WHOLESALPR;
BY DIRECTOR
BY RATING
BY COPIES
WHERE COPIES LT 3
WHERE DIRECTOR EQ 'HITCHCOCK A.'
ON COPIES SUB-TOTAL AS '*SUB: '
ON RATING RECOMPUTE AS '*REC: '
END
```

The output is:

DIRECTOR	RATING	COPIES	LISTPR	WHOLESALEPR	RATIO
HITCHCOCK A.	NR	2	19.98	9.00	2.22
*SUB:	2		19.98	9.00	2.22
*REC:	NR		19.98	9.00	2.22
	PG	2	19.98	9.00	2.22
			19.98	9.00	2.22
*SUB:	2		39.96	18.00	4.44
*REC:	PG		39.96	18.00	2.22
	PG13	2	19.98	9.00	2.22
*SUB:	2		19.98	9.00	2.2
*REC:	PG13		19.98	9.00	2.2
HITCHCOCK A.	R	2	19.98	9.00	2.2
*SUB:	2		19.98	9.00	2.2
*REC:	R		19.98	9.00	2.2
*TOTAL DIRECTOR HITCHCOCK A.			99.90	45.00	11.1
TOTAL			99.90	45.00	2.2

Example: Using Multiple Summary Commands With Prefix Operators

The following request prints the average value of LISTPR and the recomputed value of RATIO on the lines associated with sort field RATING. The SUB-TOTAL command associated with sort field COPIES is propagated to all fields on the DIRECTOR sort field lines and to the WHOLESALEPR and RATIO1 columns associated with the RATING sort field. The grand total line is suppressed for this request.

```
TABLE FILE MOVIES
PRINT LISTPR WHOLESALEPR
COMPUTE RATIO/D6.2 = LISTPR/WHOLESALEPR;
COMPUTE RATIO1/D6.2 = LISTPR/WHOLESALEPR;
BY DIRECTOR
BY RATING
BY COPIES
WHERE COPIES LT 3
  WHERE DIRECTOR EQ 'KAZAN E.'
  ON RATING RECOMPUTE AVE. LISTPR RATIO AS '*REC: '
  ON COPIES SUB-TOTAL AS '*SUB: '
  ON TABLE NOTOTAL
END
```

On the output:

- ❑ The values of WHOLESALEPR and RATIO1 on the row labeled *REC are subtotals because of propagation of the SUB-TOTAL command to the fields not specified in the RECOMPUTE command.
- ❑ The LISTPR value is an average and the value of RATIO (which has the same definition as RATIO1) is recomputed because these two fields are specified in the RECOMPUTE command.
- ❑ The SUB-TOTAL command is propagated to the DIRECTOR row.

The output is:

DIRECTOR	RATING	COPIES	LISTPR	WHOLESALEPR	RATIO	RATIO1
KAZAN E.	NR	1	24.98	14.99	1.67	1.67
*SUB:	1		24.98	14.99	1.67	1.67
		2	19.95	9.99	2.00	2.00
*SUB:	2		19.95	9.99	2.00	2.00
*REC:	NR		22.46	24.98	.90	3.66
*TOTAL	DIRECTOR KAZAN E.		44.93	24.98	3.66	3.66

Example: Propagation of Summary Commands With Field Lists

In the following request, the RECOMPUTE command has a field list.

```

SET SUMMARYLINES = OLD
TABLE FILE MOVIES
PRINT LISTPR WHOLESALEPR
COMPUTE RATIO/D6.2 = LISTPR/WHOLESALEPR;
COMPUTE RATIO1/D6.2 = LISTPR/WHOLESALEPR;
BY DIRECTOR
BY RATING
BY COPIES
WHERE COPIES LT 3
  WHERE DIRECTOR EQ 'KAZAN E.'
  ON RATING RECOMPUTE LISTPR RATIO AS '*REC: '
  ON COPIES SUB-TOTAL AS '*SUB: '
END

```

With SUMMARYLINES=OLD, only those fields have values on the report output:

DIRECTOR	RATING	COPIES	LISTPR	WHOLESALEPR	RATIO	RATIO1
KAZAN E.	NR	1	24.98	14.99	1.67	1.67
*SUB: 1			24.98		1.67	
		2	19.95	9.99	2.00	2.00
*SUB: 2			19.95		2.00	
*REC: NR			44.93		1.80	
*TOTAL DIRECTOR KAZAN E.			44.93		3.66	
TOTAL			44.93		1.80	

With SUMMARYLINES=NEW, SUB-TOTAL propagates to all of the columns that would otherwise be unpopulated. The grand total line inherits the RECOMPUTE command for the fields listed in its field list, and the SUB-TOTAL command propagates to the other columns:

DIRECTOR	RATING	COPIES	LISTPR	WHOLESALEPR	RATIO	RATIO1
KAZAN E.	NR	1	24.98	14.99	1.67	1.67
*SUB: 1			24.98	14.99	1.67	1.67
		2	19.95	9.99	2.00	2.00
*SUB: 2			19.95	9.99	2.00	2.00
*REC: NR			44.93	24.98	1.80	3.66
*TOTAL DIRECTOR KAZAN E.			44.93	24.98	3.66	3.66
TOTAL			44.93	24.98	1.80	3.66

Reference: Usage Notes for Combinations of Summary Commands

- ❑ Summary processing differs for summary commands that have prefix operators and summary lines that do not use prefix operators. If any summary command uses prefix operators, the entire request uses prefix operator processing. If processing without prefix operators is initiated, but a subsequent field requires prefix operator processing, the following message is generated and processing halts:

(FOC36376) CANNOT COMBINE SUBTOTAL/RECOMPUTE STYLES WHEN SUMMARYLINES=OLD

You can prevent this message by setting SUMMARYLINES=NEW to invoke prefix operator processing.

- ❑ SET SUMMARYLINES=EXPLICIT affects propagation of summary commands to the grand total line by making it consistent with the behavior for any sort break. Therefore, with this setting in effect, SUB-TOTAL and SUMMARIZE propagate to the grand total line but SUBTOTAL and RECOMPUTE do not.

Producing Summary Columns for Horizontal Sort Fields

How to:

Produce a Summary Operation on a Horizontal Sort Field

Reference:

Usage Notes for Summaries on ACROSS Fields

The summary commands SUBTOTAL, SUB-TOTAL, SUMMARIZE, and RECOMPUTE can be used with horizontal sort breaks.

Syntax: How to Produce a Summary Operation on a Horizontal Sort Field

```
{ACROSS|ON} breakfield [AS 'text1'] sumoption [AS 'text2']
      [COLUMNS c1 [AND c2 ...]]
```

where:

breakfield

Is the ACROSS field whose for which you want to generate the summary option. The end of the values for the ACROSS field triggers the summary operation.

sumoption

Can be one of the following: SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE.

'*text1*'

Is the column heading to use for the break field on the report output.

'*text2*'

Is the text that prints on the top of the summary column.

COLUMNS *c1*, *c2* ...

Lists the specific ACROSS values that you want to display on the report output in the order in which you want them. This list of values cannot be specified in an ON phrase. If it is specified in an ACROSS phrase, it must be the last option specified in the ACROSS phrase.

Reference: Usage Notes for Summaries on ACROSS Fields

- ❑ SUMMARIZE and SUB-TOTAL operate on the ACROSS field for which they are specified and for all higher level ACROSS fields. They do not operate on BY fields. SUBTOTAL and RECOMPUTE operate only on the ACROSS field for which they are specified.
- ❑ SUMMARIZE and SUB-TOTAL commands specified for a BY field operate on that BY and all higher level BY fields. They do not operate on ACROSS fields.

- ❑ ROW-TOTAL, ACROSS-TOTAL, SUBTOTAL, and SUB-TOTAL sum the values in the columns. Unlike SUMMARIZE and RECOMPUTE, they do not reapply calculations other than sums.
- ❑ Summary commands specified in an ON TABLE phrase operate on columns, not rows.

Example: Using Summary Commands With ACROSS

The following request sums units and dollars and calculates the unit cost by product and across region and month. The ACROSS MNTH RECOMPUTE command creates totals of units and dollars, and recomputes the calculated value for the selected months within regions. The ACROSS REGION RECOMPUTE command does the same for the selected regions. The ON TABLE SUMMARIZE command creates summary rows. It has no effect on columns:

```
DEFINE FILE GGSALES
MNTM/MTr = DATE;
END
TABLE FILE GGSALES
SUM
  UNITS/I5 AS 'UNITS'                OVER
  DOLLARS/I6 AS 'DOLLARS'            OVER
  COMPUTE DOLLPER/I6 = DOLLARS/UNITS; AS 'UNIT COST'
BY PRODUCT
ACROSS REGION RECOMPUTE AS 'Region Sum' COLUMNS 'Northeast' AND 'West'
ACROSS MNTH RECOMPUTE AS 'Month Sum' COLUMNS 'November' AND 'December'
WHERE DATE FROM '19971101' TO '19971231';
WHERE PRODUCT EQ 'Capuccino' OR 'Espresso';
ON TABLE SUMMARIZE AS 'Grand Total'
  ON TABLE HOLD FORMAT HTML
END
```

The output is:

PAGE 1								
		Region						
		Northeast			West			Region Sum
		MNTH						
		NOVEMBER	DECEMBER	Month Sum	NOVEMBER	DECEMBER	Month Sum	
Product								
Capuccino	UNITS	2282	1188	3470	2535	4051	6586	10056
	DOLLARS	25994	13668	39662	31153	57421	88574	128236
	UNIT COST	11	11	11	12	14	13	12
Espresso	UNITS	1947	2403	4350	3088	3732	6820	11170
	DOLLARS	23629	30605	54234	36123	51400	87523	141757
	UNIT COST	12	12	12	11	13	12	12
Grand Total	UNITS	4229	3591	7820	5623	7783	13406	21226
	DOLLARS	49623	44273	93896	67276	108821	176097	269993
	UNIT COST	11	12	12	11	13	13	12

Performing Calculations at Sort Field Breaks

How to:

Use Subtotals in Calculations

Reference:

Usage Notes for RECAP and COMPUTE

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.

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, as described in [Using Expressions](#) on page 323. You must end the expression with a semicolon.

WHEN *expression*

Is for use with RECAP only. It specifies the conditional display of RECAP lines as determined by a Boolean expression (see [Conditionally Displaying Summary Lines and Text](#) on page 321). 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.
- ❑ The calculations in a RECAP or COMPUTE can appear anywhere under the control break, along with any text. (For details, see .
- ❑ In an ON phrase, a COMPUTE command is the same as a RECAP command.

- 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:

DEPARTMENT	PAY_DATE	DED_AMT	GROSS
-----	-----	-----	-----
MIS	81/11/30	\$1,406.79	\$2,147.75
	81/12/31	\$1,406.79	\$2,147.75
	82/01/29	\$1,740.89	\$3,247.75
	82/02/26	\$1,740.89	\$3,247.75
	82/03/31	\$1,740.89	\$3,247.75
	82/04/30	\$3,386.73	\$5,890.84
	82/05/28	\$3,954.35	\$6,649.50
	82/06/30	\$4,117.03	\$7,460.00
	82/07/30	\$4,117.03	\$7,460.00
	82/08/31	\$4,575.72	\$9,000.00
** DEPT_NET		\$22,311.98	
PRODUCTION	81/11/30	\$141.66	\$833.33
	81/12/31	\$141.66	\$833.33
	82/01/29	\$1,560.09	\$3,705.84
	82/02/26	\$2,061.69	\$4,959.84
	82/03/31	\$2,061.69	\$4,959.84
	82/04/30	\$2,061.69	\$4,959.84
	82/05/28	\$3,483.88	\$7,048.84
	82/06/30	\$3,483.88	\$7,048.84
	82/07/30	\$3,483.88	\$7,048.84
	82/08/31	\$4,911.12	\$9,523.84
** DEPT_NET		\$27,531.14	

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 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

How to:

Suppress Grand Totals

You can use the NOTOTAL command to suppress grand totals in 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 280.

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
```

Conditionally Displaying Summary Lines and Text

The output is:

EMP_ID	BANK_ACCT	CURR_SAL	GROSS	DED_AMT
-----	-----	-----	-----	-----
117593129	40950036	\$18,480.00	\$6,099.50	\$2,866.18
*TOTAL	40950036	\$18,480.00	\$6,099.50	\$2,866.18
*TOTAL	117593129	\$18,480.00	\$6,099.50	\$2,866.18
119329144	160633	\$29,700.00	\$2,475.00	\$1,427.24
*TOTAL	160633	\$29,700.00	\$2,475.00	\$1,427.24
*TOTAL	119329144	\$29,700.00	\$2,475.00	\$1,427.24
123764317	819000702	\$26,862.00	\$17,094.00	\$11,949.44
*TOTAL	819000702	\$26,862.00	\$17,094.00	\$11,949.44
*TOTAL	123764317	\$26,862.00	\$17,094.00	\$11,949.44
326179357	122850108	\$21,780.00	\$9,075.00	\$6,307.00
*TOTAL	122850108	\$21,780.00	\$9,075.00	\$6,307.00
*TOTAL	326179357	\$21,780.00	\$9,075.00	\$6,307.00
451123478	136500120	\$16,100.00	\$9,130.00	\$3,593.92
*TOTAL	136500120	\$16,100.00	\$9,130.00	\$3,593.92
*TOTAL	451123478	\$16,100.00	\$9,130.00	\$3,593.92
818692173	163800144	\$27,062.00	\$22,013.75	\$15,377.40
*TOTAL	163800144	\$27,062.00	\$22,013.75	\$15,377.40
*TOTAL	818692173	\$27,062.00	\$22,013.75	\$15,377.40

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 SUBFOOT, SUBHEAD, SUBTOTAL, SUBTOTAL, SUMMARIZE, RECOMPUTE, and RECAP. For complete details on using the WHEN phrase, see [Conditionally Formatting Reports With the WHEN Clause](#) on page 411.

Example: Conditionally Displaying Summary Lines and Text

In a sales report that covers four regions (Midwest, Northeast, Southeast, and West), you may only want to display a subtotal when total dollar sales are greater than \$11,500,000. The following request accomplishes this by including criteria that trigger the display of a subtotal when dollar sales exceed \$11,500,000 and subfooting text when dollar sales are less than \$11,500,000.

```
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	4379994
	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	4473517
	Food	340234	4202337
	Gifts	235042	2977092
*TOTAL West		932039	11652946
TOTAL		3688991	46156290

8 | Using Expressions

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 amp variable, or use it in screening. You can combine simpler ones to build increasingly complex expressions.

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.

Topics:

- ❑ Using Expressions in Commands and Phrases
- ❑ Types of Expressions
- ❑ Creating a Numeric Expression
- ❑ Creating a Date Expression
- ❑ Creating a Date-Time Expression
- ❑ Creating a Character Expression
- ❑ Creating a Variable Length Character Expression
- ❑ Creating a Logical Expression
- ❑ Creating a Conditional Expression

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 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

In this section:

Expressions and Field Formats

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 327.

- ❑ **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 Expression](#) on page 331.
- ❑ 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 on these functions, see the *Using Functions* 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 344.

Note: Text fields can be assigned to alphanumeric fields and receive assignment from alphanumeric fields. Text fields can also participate in expressions using the operators CONTAINS and OMITs.

- ❑ **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 351.
- ❑ **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 353.

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.

Example: Assigning a Field Format of Sufficient Length

The following example contains a character expression that concatenates a 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

In this section:

Order of Evaluation

How to:

Express a Number in Scientific Notation

Reference:

Arithmetic Operators

A numeric expression performs a calculation that uses numeric constants, fields, operators, and 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 constant in scientific notation. For example:

```
COMPUTE COST/D12.2 = EXPN(8E+3) ;
```

For syntax usage, see [How to Express a Number in Scientific Notation](#) on page 328.

- ❑ 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 329.

- ❑ 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 330.

Before they are used in calculations, numeric values are generally 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 difference in rounding. .

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 zeros display for the field value.

Note: You can change the overflow character by issuing the SET OVERFLOWCHAR command.

For detailed information on rounding behavior for numeric data formats, see the *Describing Data* manual.

Syntax: **How to Express a Number in Scientific Notation**

In an IF clause, use the following:

```
IF field op n[.nn]{E|D|e|d}[±|-]p
```

In a WHERE clause, use the following:

```
WHERE field op EXPN(n[.nn]{E|D|e|d}[±|-]p);
```

In a COMPUTE command, use the following:

```
COMPUTE field[/format] = EXPN(n[.nn]{E|D|e|d}[±|-]p);
```

In a DEFINE command, use the following:

```
DEFINE FILE filename  
field[/format] = EXPN(n[.nn]{E|D|e|d}[±|-]p);  
END
```

In a DEFINE in the Master File, use the following:

```
DEFINE field[/format] = EXPN(n[.nn]{E|D|e|d}[±|-]p);
```

where:

field

Is a field in a request.

/format

Is the optional format of the field. For information on formats, see the *Describing Data* manual.

op

Is a relational operator in a request.

n.nn

Is a numeric constant that consists of a whole number component, followed by a decimal point, followed by a fractional component.

E, D, e, d

Denotes scientific notation. E, e, d, and D are interchangeable.

±, -

Indicates if *p* is positive or negative. Positive is the default.

p

Is the power of 10 to which to raise the number. The range of values for *p* is between -78 and +78.

Note: EXPN is useful for calculations on fields with F and D formats. It is generally not useful for calculations on fields with P or I formats.

Example: Evaluating a Number in Scientific Notation

You can use scientific notation in an IF or WHERE clause to express 8000 as 8E+03:

```
IF RCOST LT 8E+03
WHERE RCOST LT EXPN(8E+03)
```

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.

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	9.79

Creating a Date Expression

In this section:

- Formats for Date Values
- Performing Calculations on Dates
- Cross-Century Dates With DEFINE and COMPUTE
- Returned Field Format Selection
- Using a Date Constant in an Expression
- Extracting a Date Component
- Combining Fields With Different Formats in an Expression

A date 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';
```

This requires single quotation marks around the date constant.

- ❑ 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 hire date and the date of his first salary increase:

```
COMPUTE DIFF/I4 = YMD (HIRE_DATE, FST.DAT_INC);
```

Formats for Date Values

Reference:

Base Dates for Date Formats

Impact of Date Formats on Storage and Display

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.

- ❑ 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, YYMD, MDYY, DMY, MDY, and DMY	1900/12/31
YM, YYM, MYY, and MY	1901/01 on z/OS and VM 1900/12/31 on Windows and UNIX
YQ, YYQ, QYY, and QY	1901 Q1
JUL and YYJUL	1900/365
D M Y, YY Q W	There is no base date for these formats; these are just numbers, not dates.

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:

Value	Date Format (For example: MDYY)		Integer, Packed, Decimal, or Alphanumeric Format (For example: A8MDYY)	
	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 an 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, the 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.

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 022899 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 Date-Time Expression**In this section:**

Specifying a Date-Time Value

Manipulating Date-Time Values

How to:

Specify the Order of Date Components in a Date-Time Field

A *date-time* expression returns date and time components. You can create these expressions using a variety of supplied date-time functions. For details about date-time functions, see the *Using Functions* manual.

Syntax: How to Specify the Order of Date Components in a Date-Time Field

```
SET DATEFORMAT = option
```

where:

option

Can be one of the following: MDY, DMY, YMD, or MYD. MDY is the default value for the U.S. English format.

For an example, see [Specifying the Order of Date Components for a Date-Time Field](#) on page 340.

Specifying a Date-Time Value

An external date-time value is a constant in character format from one of the following sources:

- A sequential data source.
- Typed by an application user at a terminal or workstation.
- Used in an expression in a WHERE, IF, DEFINE, or a COMPUTE.

A date-time constant typed by an application user at a terminal or workstation, or a date-time value as it appears in a character file has one of the following formats:

```
time_string [date_string]
date_string [time_string]
```

A date-time constant in a COMPUTE, DEFINE, or WHERE expression must have one of the following formats:

```
DT(time_string [date_string])
DT(date_string [time_string])
```

A date-time constant in an IF expression has one of the following formats:

```
'time_string [date_string]'
'date_string [time_string]'
```

If the value contains no blanks or special characters, the single quotation marks are not necessary. Note that the DT prefix is not supported in IF criteria.

where:

time_string

Cannot contain blanks. Time components are separated by colons, and may be followed by AM, PM, am, or pm. For example:

```
14:30:20:99      (99 milliseconds)
14:30
14:30:20.99      (99/100 seconds)
14:30:20.999999  (999999 microseconds)
02:30:20:500pm
```

Note that the second can be expressed with a decimal point or be followed by a colon:

- ❑ If there is a colon after the second, the value following it represents the millisecond. There is no way to express the microsecond or nanosecond using this notation.
- ❑ A decimal point in the second value indicates the decimal fraction of a second. A microsecond can be represented using six decimal digits. A nanosecond can be represented using nine decimal digits.

date_string

Can have one of the following three formats:

- ❑ **Numeric string format** is exactly four, six, or eight digits. Four-digit strings are considered to be a year (century must be specified). The month and day are set to January 1. Six and eight-digit strings contain two or four digits for the year, followed by two for the month, and then two for the day. Because the component order is fixed with this format, the DATEFORMAT setting described in [How to Specify the Order of Date Components in a Date-Time Field](#) on page 337 is ignored.

If a numeric-string format longer than eight digits is encountered, it is treated as a combined date-time string in the Hn format. The following are examples of numeric string date constants:

```
99
1999
19990201
```

- ❑ **Formatted-string format** contains a one or two-digit day, a one or two-digit month, and a two or four-digit year separated by spaces, slashes, hyphens, or periods. All three parts must be present and follow the DATEFORMAT setting described in [How to Specify the Order of Date Components in a Date-Time Field](#) on page 337. If any of the three fields is four digits, it is interpreted as the year, and the other two fields must follow the order given by the DATEFORMAT setting. The following are examples of formatted-string date constants:

```
1999/05/20
5 20 1999
99.05.20
1999-05-20
```

- ❑ **Translated-string format** contains the full or abbreviated month name. The year must also be present in four-digit or two-digit form. If the day is missing, day 1 of the month is assumed; if present, it can have one or two digits. If the string contains both a two-digit year and a two-digit day, they must be in the order given by the DATEFORMAT setting. For example:

```
January 6 2000
```

Note:

- ❑ The date and time strings must be separated by at least one blank space. Blank spaces are also permitted at the beginning and end of the date-time string or immediately before an am/pm indicator.
- ❑ In each date format, two-digit years are interpreted using the [F]DEFCENT and [F]YRTHRESH settings.

Example: Assigning Date-Time Literals

The DT prefix can be used in a COMPUTE, DEFINE, or WHERE expression to assign a date-time literal to a date-time field. For example:

```
DT2/HYYMDS = DT(20051226 05:45);
DT3/HYYMDS = DT(2005 DEC 26 05:45);
DT4/HYYMDS = DT(December 26 2005 05:45);
```

Example: Specifying the Order of Date Components for a Date-Time Field

The following request sets DATEFORMAT to MYD:

```
SET DATEFORMAT = MYD
DEFINE FILE EMPLOYEE
DTFLDYMD/HYYMDI = DT(APR 04 05);
END

TABLE FILE EMPLOYEE
PRINT CURR_SAL DTFLDYMD
END
```

The output shows that the natural date literal 'APR 04 05' is interpreted as April 5, 1904:

CURR_SAL	DTFLDYMD
-----	-----
\$11,000.00	1904/04/05 00:00
\$13,200.00	1904/04/05 00:00
\$18,480.00	1904/04/05 00:00
\$9,500.00	1904/04/05 00:00
\$29,700.00	1904/04/05 00:00
\$26,862.00	1904/04/05 00:00
\$21,120.00	1904/04/05 00:00
\$18,480.00	1904/04/05 00:00
\$21,780.00	1904/04/05 00:00
\$16,100.00	1904/04/05 00:00
\$9,000.00	1904/04/05 00:00
\$27,062.00	1904/04/05 00:00

Example: Reading Date-Time Values From a Transaction File

The DATTRANS comma-delimited transaction file has an ID field and a date-time field that contains both the date (as eight characters) and time (in the format hour:minute:second):

```
01, 20000101 02:57:25,$
02, 19991231 14:05:35,$
```

Because the transaction file contains the dates in numeric string format, the DATEFORMAT setting is not used, and the dates are entered in YMD order.

The following transaction file is also valid. It contains formatted string dates that comply with the default DATEFORMAT setting, MDY:

```
01, 01/01/2000 02:57:25,$
02, 12/31/1999 14:05:35,$
```

The following Master File describes the FOCUS data source named DATETIME, which receives these values:

```
FILE=DATETIME, SUFFIX=FOC , $
SEGNAME=DATETIME, SEGTYPE=S0 , $
FIELD=ID, ID, USAGE = I2 , $
FIELD=DT1, DT1, USAGE=HYYMDS , $
```

Example: Using a Date-Time Value in a COMPUTE Command

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME AND COMPUTE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
WHERE CURR_JOBCODE LIKE 'B%'
END
```

The output is:

LAST_NAME	FIRST_NAME	NEWSAL	RAISETIME
SMITH	MARY	\$14,520.00	2000/01/01 9:00AM
JONES	DIANE	\$20,328.00	2000/01/01 9:00AM
ROMANS	ANTHONY	\$23,232.00	2000/01/01 9:00AM
MCCOY	JOHN	\$20,328.00	2000/01/01 9:00AM
BLACKWOOD	ROSEMARIE	\$23,958.00	2000/01/01 9:00AM
MCKNIGHT	ROGER	\$17,710.00	2000/01/01 9:00AM

Example: Using a Date-Time Value in WHERE Criteria

In a WHERE clause, a date-time constant must use the DT() format:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE
WHERE TRANSDATE GT DT(2000/01/01 02:57:25)
END
```

The output is:

CUSTID	TRANSDATE
1118	2000/06/26 05:45
1237	2000/02/05 03:30

Example: Using a Date-Time Value in IF Criteria

In an IF clause, a date-time constant must be enclosed in single quotation marks if it contains any blanks:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE
IF TRANSDATE GT '2000/01/01 02:57:25'
END
```

Note: The DT prefix for a date-time constant is not supported in an IF clause.

The output is:

CUSTID	TRANSDATE
1118	2000/06/26 05:45
1237	2000/02/05 03:30

Example: Specifying Universal Date-Time Input Values

With DTSTANDARD settings of STANDARD and STANDARDU, the following date-time values can be read as input:

Input Value	Description
14:30[:20,99]	Comma separates time components instead of period
14:30[:20.99]Z	Universal time
15:30[:20,99]+01 15:30[:20,99]+0100 15:30[:20,99]+01:00	Each of these is the same as above in Central European Time
09:30[:20.99]-05	Same as above in Eastern Standard Time

Note that these values are stored identically internally with the STANDARDU setting. With the STANDARD setting, everything following the Z, +, or - is ignored.

Manipulating Date-Time Values

The only direct operations that can be performed on date-time variables and constants are comparison using a logical expression, and simple assignment of the form A = B.

Computations only allow direct assignment within data types: alpha to alpha, numeric to numeric, date to date, and date-time to date-time. All other operations are accomplished through a set of date-time functions.

Any two date-time values can be compared, even if their lengths do not match.

If a date-time field supports missing values, fields that contain the missing value have a greater value than any date-time field can have. Therefore, in order to exclude missing values from the report output when using a GT or GE operator in a selection test, it is recommended that you add the additional constraint field NE MISSING to the selection test:

```
date_time_field {GT|GE} date_time_value AND date_time_field NE MISSING
```

Assignments are permitted between date-time formats of equal or different lengths. Assigning a 10-byte date-time value to an 8-byte date-time value truncates the microsecond portion (no rounding takes place). Assigning a short value to a long one sets the low-order three digits of the microseconds to zero.

Other operations, including arithmetic, concatenation, EDIT, and LIKE on date-time operands are not supported. Prefix operators that work with alphanumeric fields are supported.

Example: Testing for Missing Date-Time Values

Consider the DATETIM2 Master File:

```
FILE=DATETIM2, SUFFIX=FOC , $
SEGNAME=DATETIME, SEGTYPE=S0 , $
FIELD=ID, ID, USAGE = I2 , $
FIELD=DT1, DT1, USAGE=HYMDS, MISSING=ON, $
```

Field DT1 supports missing values. Consider the following request:

```
TABLE FILE DATETIM2
PRINT ID DT1
END
```

The resulting report output shows that in the instance with ID=3, the field DT1 has a missing value:

```
ID  DT1
--  ---
 1  2000/01/01 02:57:25
 2  1999/12/31 00:00:00
 3  .
```

The following request selects values of DT1 that are greater than 2000/01/01 00:00:00 and are not missing:

```
TABLE FILE DATETIM2
PRINT ID DT1
WHERE DT1 NE MISSING AND DT1 GT DT(2000/01/01 00:00:00);
END
```

The missing value is not included in the report output:

```
ID  DT1
--  ---
 1  2000/01/01 02:57:25
```

Example: Assigning a Different Usage Format to a Date-Time Column

Consider the following request using the VIDEOTR2 data source:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AND COMPUTE
DT2/HYMDH = TRANSDATE;
T1/HHIS = TRANSDATE;
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  TRANSDATE          DT2          T1
-----  -----
1118    2000/06/26 05:45    2000/06/26 05    05:45:00
1237    2000/02/05 03:30    2000/02/05 03    03:30:00
```

Creating a Character Expression

In this section:

- Embedding a Quotation Mark in a Quote-Delimited Literal String
- Concatenating Character Strings

A character expression uses alphanumeric constants, fields, concatenation operators, or functions to derive an alphanumeric value.

Both text and alphanumeric fields can be assigned values stored in text fields or alphanumeric expressions in TABLE COMPUTE, MODIFY COMPUTE, and DEFINE commands. If an alphanumeric field is assigned the value of a text field that is too long for the alphanumeric field, the value is truncated before being assigned to the alphanumeric field.

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
```

- ❑ A text field.

Note: Non-printable characters are not supported in an alphanumeric constant.

Embedding a Quotation Mark in a Quote-Delimited Literal String

Under certain conditions, you can use quote-delimited strings containing embedded quotation marks. Within the string, you can use either one single quotation mark or two contiguous single quotation marks to represent the single quotation mark. Both are interpreted as a single quotation mark.

You can use quote-delimited strings in the following instances:

- ❑ 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. This 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 the end of a 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 Variable Length Character Expression

In this section:

- Using Concatenation With AnV Fields
- Using the EDIT Function With AnV Fields
- Using CONTAINS and OMITS With AnV Fields
- Using LIKE With AnV Fields
- Using the EQ, NE, LT, GT, LE, and GE Operators With AnV Fields
- Using the DECODE Function With AnV Fields
- Using the Assignment Operator With AnV Fields

As an alphanumeric type, an AnV field can be used in arithmetic and logical expressions in the same way that the An type is used.

- ❑ An expression that contains AnV type fields can be of either the AnV or An type.
- ❑ The type that results from the expression depends on the specific type of operation, as described in subsequent sections.

Note: Because AnV fields have two bytes of overhead and there is additional processing required to strip them, AnV format is not recommended for use in non-relational data sources.

Using Concatenation With AnV Fields

If either of the operands in a concatenation between two fields is an AnV field, variable length alphanumeric rules are used to perform the concatenation:

- ❑ The size of the concatenated string is the sum of the sizes of the operands.
- ❑ For weak concatenation, the actual length of the concatenated string is the sum of the two actual lengths of the input strings.
- ❑ For strong concatenation, the actual length stored in an AnV field of the concatenated string is the sum of the actual length of the first input string minus its number of trailing blanks plus the actual length of the second string.
- ❑ For any An field in the concatenation, the size and length are equal.

Using the EDIT Function With AnV Fields

The following expression results in an AnV format only when x has AnV format.

`EDIT(x,mask)`

The actual length of the result is the number of characters in *mask* other than '\$'.

Note that an actual length of zero may result.

EDIT(x) can be used to convert an AnV field to an integer value when x has AnV format.

Using CONTAINS and OMITS With AnV Fields

The only difference in evaluation of the CONTAINS and OMITS operators with AnV fields occurs when one of the operands has an actual length of zero.

In the following examples, the field Z has an actual length of zero, but X and Y do not:

Expression	Result
<code>Z CONTAINS Y</code>	FALSE
<code>X CONTAINS Z</code>	TRUE
<code>Z CONTAINS Z</code>	TRUE
<code>Z OMITS Y</code>	TRUE
<code>X OMITS Z</code>	FALSE
<code>Z OMITS Z</code>	FALSE

Using LIKE With AnV Fields

The only difference in evaluation of the following expression occurs when x has an actual length of zero:

`x LIKE mask ...`

In the following example, the field instance Z has an actual length of zero:

`Z LIKE mask ...`

This expression evaluates to TRUE only when the mask consists exclusively of percent ('%') signs.

Note that no other mask can evaluate to an empty string. Even the mask in the following expression has a length of one, and therefore the expression evaluates as FALSE:

```
Z LIKE ''
```

Using the EQ, NE, LT, GT, LE, and GE Operators With AnV Fields

As with An type fields, operations are evaluated on the assumption that the shorter operand is padded with blanks.

Therefore, even an empty AnV field, Z, is compared as a field consisting of all blanks.

In the following examples, Z is an empty AnV field instance and X is an AnV field instance that is not empty and contains non-blank characters:

Expression	Result
Z EQ Z Z GE Z Z LE Z	TRUE
Z NE Z Z LT Z Z GT Z	FALSE
Z EQ X	FALSE
Z NE X	TRUE
Z LT X	TRUE
Z GT X	FALSE
Z LE X	TRUE
Z GE X	FALSE
X EQ Z	FALSE
X NE Z	TRUE
X LT Z	FALSE
X GT Z	TRUE
X LE Z	FALSE

Expression	Result
X GE Z	TRUE

Using the DECODE Function With AnV Fields

`DECODE alphafield (value 'result'...`

The use of either an *An* or *AnV* field with DECODE causes a result of type *An* as long as the *result* part of the value-result pairs is provided as a constant. (Constants are type *An*.)

Using the Assignment Operator With AnV Fields

There are three situations to consider when using the assignment operator with the *AnV* format: *AnV* data type on the right hand side only, *AnV* data type on both sides, and *AnV* data type on the left side only.

`fld/An = AnV_type_expression;`

- ❑ The actual length of the evaluated expression is lost on assignment to the *An* field.
- ❑ The size of the *AnV* result does not prevent assignment to a shorter *An* format field:
 - ❑ If the result of the expression has an actual length that is shorter than the length of the field on the left side of the assignment operator, the result is padded with blanks.
 - ❑ If the result of the expression has an actual length that is longer than the length of the field on the left side of the assignment operator, the result is truncated.

`fld/AnV = AnV_type_expression;`

- ❑ The length of the result is assigned as the length of the field on the left of the assignment operator unless it exceeds the field's declared size. In this case, the length assigned is the declared size (*n*).

- ❑ The size of the AnV evaluation result does not prevent assignment to a shorter AnV field:
 - ❑ If the length of the result of the expression is shorter than the size of the field on the left side of the assignment operator, the result is padded with blanks.
 - ❑ If the result of the expression has an actual length that is longer than the size of the field on the left side of the assignment operator, the result is truncated.

fld/AnV = An_type_expression;

- ❑ The length of the field on the left side of the assignment operator is assigned equal to its size (n).
- ❑ The actual length of the result is verified against the size n declared for the AnV field. An error is generated if the result is longer than n .

Creating a Logical Expression

How to:

Write a Relational Expression

Write a Boolean Expression

Reference:

Logical Operators

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 [Selecting Records for Your Report](#) on page 157.

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.
IS MISSING	Returns the value TRUE if the field is missing .
IS-NOT MISSING	Returns the value TRUE if the field is not missing.

Syntax: How to Write a Relational Expression

Any of the following are valid for a relational expression:

```
value {EQ|NE} value value {LE|LT} value value {GE|GT} 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 is 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**How to:**

Write 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.

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.

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:

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 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:

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.

9 Customizing Tabular 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.

If you save your report output in HTML, Excel 2000, PDF, or PostScript format, you have many additional formatting options that are described in [Styling Reports](#) on page 491.

Topics:

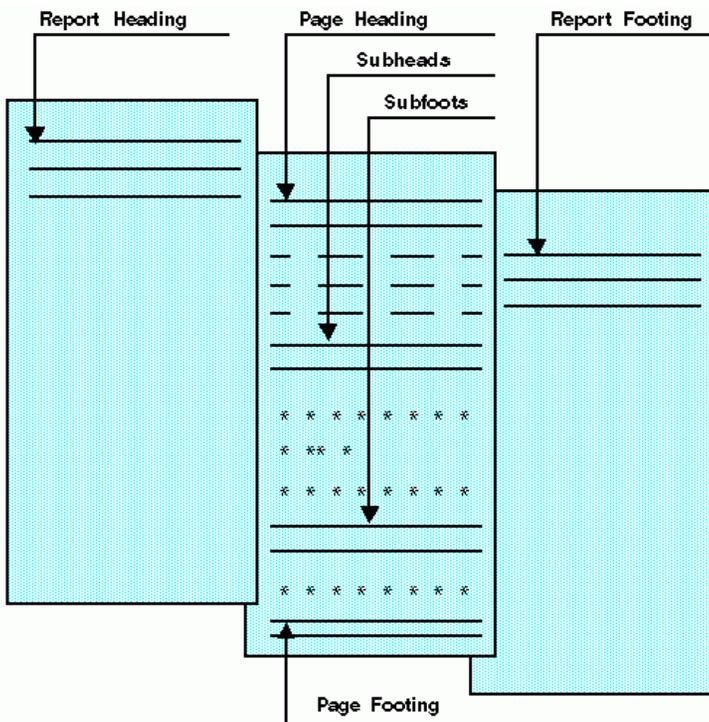
- ❑ Producing Headings and Footings
- ❑ Creating Paging and Numbering
- ❑ Suppressing Fields: SUP-PRINT or NOPRINT
- ❑ Reducing a Report's Width: FOLD-LINE and OVER
- ❑ Positioning Columns: IN
- ❑ Separating Sections of a Report: SKIP-LINE and UNDER-LINE
- ❑ Controlling Column Spacing: SET SPACES
- ❑ Creating New Column Titles: AS
- ❑ Customizing Column Names: SET QUALTITLES
- ❑ Column Title Justification
- ❑ Customizing Reports With SET Parameters
- ❑ Conditionally Formatting Reports With the WHEN Clause
- ❑ Controlling the Display of Empty Reports

Producing Headings and Footings

In this section:

- Limits for Headings and Footings
- Report and Page Headings
- Report and Page Footings
- Subheads and Subfoots
- Using Data in Headings and Footings
- Positioning Text
- Extending Heading and Footing Code to Multiple Lines
- Producing a Free-Form Report

You can use a variety of headings and footings to clarify the information presented in your reports. The following diagram illustrates the available options:



Limits for Headings and Footings

The following limitations apply to report headings and footings, page headings and footings, and sort headings and footings:

- ❑ In a single report, there can be a maximum of 32K characters for all types of heading and footing text.
- ❑ The maximum number of sort headings plus sort footings in one request is 64.
- ❑ The maximum limit of nested headings is 64.
- ❑ If your code for a single heading or footing line is broken into multiple lines in the report request, you can indicate that they are all a single line of heading using the <OX spot marker. For more information, see [Customizing Reports With SET Parameters](#) on page 410.
- ❑ The maximum number of objects per line in a heading or footing is 128.
- ❑ For PDF and Postscript reports, the heading or footing lines must fit within the maximum report width to be displayed properly. Also, in order for the report body to be displayed, the number of heading or footing lines must leave room on the page for at least one detail line (including column titles).

Report and Page Headings

How to:

Create a Report Heading

Create a Page Heading

A report heading is text that appears at the top of the first page of a report. A page heading is text that appears at 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 end quotation mark of the heading text is omitted, all subsequent lines of the request are 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 appears as a heading. 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 appears 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 [Using Data in Headings and Footings](#) on page 372.

Example: Creating a Report Heading

The following request creates a report heading:

```
TABLE FILE EMPLOYEE
SUM GROSS
BY DEPARTMENT BY HIGHEST PAY_DATE
ON TABLE PAGE-BREAK AND SUBHEAD
"PLEASE RETURN THIS TO MARY SMITH"
END
```

The first two pages of output follow (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
```

Example: Creating a Page Heading

The following request prints a heading on each page:

```
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

The following request creates a two-line report heading:

```
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:

```
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 <OX at the end of the line to be continued. For more information, see [Positioning Text](#) on page 374.

Report and Page Footings

How to:

Create a Report Footing

Create a Page Footing

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 between quotation marks in a report request.

Note: If the ending quotation mark of the footing text is omitted, all subsequent lines of the request are 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.

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 appears on every page.

Example: Creating a Page Footing

The following request creates a page footing:

```
TABLE FILE CAR
WRITE SALES BY COUNTRY
FOOTING
"THIS IS HOW A FOOTNOTE IS ADDED TO EACH"
"PRINTED PAGE"
END
```

The output is:

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 and Subfoots

How to:

Create a Subhead

Create Subfoots

Reference:

Usage Notes for Creating Subfoots

A subhead is text that can be placed before the sort field values change. A subfoot is text that can be placed after the sort field values change. You can use NEWPAGE on a subheading or subfooting to start a new page after the subheading or before the subfooting. It separates the subheading or subfooting from its associated data but does not separate the data from the next subheading or prior subfooting.

In conjunction with the PAGE-BREAK command, this enables you to create a cover page for each section of a report.

Note:

- ❑ If the ending quotation mark of the subheading text is omitted, all subsequent lines of the request are treated as part of the subheading.
- ❑ If the ending quotation mark of the subfooting text is omitted, all subsequent lines of the request are treated as part of the subfooting.

By default, FOCUS generates a blank line before a subheading or subfooting. You can eliminate these automatic blank lines by issuing the SET DROPBLNKLINE=ON command. For more information, see the *Developing Applications* manual.

Syntax: How to Create a Subhead

```
{ON|BY} fieldname SUBHEAD [NEWPAGE]
"text"
[WHEN expression;
```

where:

fieldname

Is the sort field before which the text is inserted.

NEWPAGE

Inserts a new page after the sort heading or before the sort footing. Column titles appear on every page. In HTML reports, blank space is added instead of a new page.

text

Is the text you supply between double quotation marks that is printed following the SUBHEAD phrase.

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

This request creates a subheading whenever the PROD_NAME field changes:

```
TABLE FILE PROD
SUM PACKAGE AND UNIT_COST
BY PROD_NAME NOPRINT BY PROD_CODE
ON PROD_NAME SUBHEAD
" SUMMARY FOR <PROD_NAME"
END
```

The output is:

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

Syntax: How to Create Subfoots

The syntax is:

```
{ON|BY} fieldname SUBFOOT [WITHIN] [MULTILINES] [NEWPAGE]
" text "
[WHEN expression ;]
```

where:

fieldname

Is the field after which the text is inserted.

WITHIN

Causes the fields in the SUBFOOT to be calculated within each value of *fieldname*. Without this option, a field in the SUBFOOT is taken from the last line of report output above the subfooting.

text

Is the text you supply between double quotation marks that is printed following the SUBFOOT phrase.

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 [Including Totals and Subtotals](#) on page 269.

NEWPAGE

Inserts a new page after the heading or before the footing. Column titles appear on every page. In HTML reports, blank space is added instead of a new page.

WHEN *expression*

Specifies a conditional subfoot in the printing of a report, as determined by a Boolean expression. 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.

- ❑ 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 372.
- ❑ 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.
- ❑ SUBFOOT WITHIN is useful where a prefixed field within a sort break would result in a single value (for example, AVE., MIN., MAX). Use of PCT. or APCT. displays only the last value from the sort group.
- ❑ SUBFOOT WITHIN "<prefix.fieldname " does not result in the same value as SUBTOTAL prefix. The SUBFOOT WITHIN creates a display field that operates on the original input records. SUBTOTAL with a prefix operates on the internal matrix (so AVE. is the average of the SUMS or, if a display field had the prefix AVE., the average of the averages). SUBFOOT WITHIN "<AVE.field " generates an overall average.
- ❑ Prefix operators are not supported on alphanumeric fields in a WITHIN phrase.
- ❑ ST. is not supported in a SUBFOOT WITHIN phrase.

Example: Using Subfoots

This example creates a subfooting whenever the DEPARTMENT value changes:

```
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
```

The output is:

DEPARTMENT	PAY_DATE	DED_AMT	GROSS
-----	-----	-----	-----
MIS	82/08/31	\$4,575.72	\$9,000.00
	82/07/30	\$4,117.03	\$7,460.00
	82/06/30	\$4,117.03	\$7,460.00
	82/05/28	\$3,954.35	\$6,649.50
	82/04/30	\$3,386.73	\$5,890.84
	82/03/31	\$1,740.89	\$3,247.75
	82/02/26	\$1,740.89	\$3,247.75
	82/01/29	\$1,740.89	\$3,247.75
	81/12/31	\$1,406.79	\$2,147.75
	81/11/30	\$1,406.79	\$2,147.75
DEPARTMENT	NET = 22,311.98		
PRODUCTION	82/08/31	\$4,911.12	\$9,523.84
	82/07/30	\$3,483.88	\$7,048.84
	82/06/30	\$3,483.88	\$7,048.84
	82/05/28	\$3,483.88	\$7,048.84
	82/04/30	\$2,061.69	\$4,959.84
	82/03/31	\$2,061.69	\$4,959.84
	82/02/26	\$2,061.69	\$4,959.84
	82/01/29	\$1,560.09	\$3,705.84
	81/12/31	\$141.66	\$833.33
	81/11/30	\$141.66	\$833.33
DEPARTMENT	NET = 27,531.14		

Example: Generating a Subfoot Within a Sort Group

The following request displays the average and minimum salary values first within department, then within department and job class, and last within department, job class and employee ID. Subfootings are generated on the department and jobcode sort fields. The DEFINE FILE command created two additional fields with the SALARY value, one for each sort break:

```

DEFINE FILE EMPDATA
SALDEPT/D6 WITH SALARY = SALARY;
SALDEPTJOB/D6 WITH SALARY = SALARY;
DEPT/A4 WITH SALARY = EDIT(DEPT, '9999');
JOB/A8 WITH SALARY = JOBCLASS;
END
TABLE FILE EMPDATA
SUM AVE.SALDEPT AS 'DEPT,AVE'
MIN.SALDEPT AS 'DEPT,MIN'
BY DEPT
SUM AVE.SALDEPTJOB AS 'DEPT/JOB, AVE' IN 32
MIN.SALDEPTJOB AS 'DEPT/JOB, MIN' IN 42
BY DEPT
BY JOB
PRINT AVE.SALARY/D6 AS 'AVE' IN 52
MIN.SALARY/D6 AS 'MIN' IN 61
BY DEPT
BY JOB
BY PIN NOPRINT
ON DEPT SUBFOOT
"*****DEPARTMENT <DEPT SUBFOOT*****"
"NOT WITHIN: AVE=<AVE.SALARY MIN=<MIN.SALARY "
ON DEPT SUBFOOT WITHIN
" WITHIN: AVE=<AVE.SALARY MIN=<MIN.SALARY "
"*****"
ON JOB SUBFOOT
"</1 *****DEPARTMENT <DEPT / JOB <JOB SUBFOOT*****"
"NOT WITHIN: AVE=<AVE.SALARY MIN=<MIN.SALARY "
ON JOB SUBFOOT WITHIN
" WITHIN: AVE=<AVE.SALARY MIN=<MIN.SALARY "
"***** </1"
WHERE DEPT EQ 'MARK'
WHERE JOBCLASS EQ '257PSB' OR '257PTB'
END

```

The report output shows that each SUBFOOT without the WITHIN phrase uses the report line above the subfooting in the calculations. The SUBFOOT within both department and jobcode uses the calculations that were specified in the second SUM command (by department by jobcode), and the SUBFOOT within department only uses the calculations that were specified in the first SUM command (by department):

DEPT	DEPT AVE	DEPT MIN	JOB	DEPT/JOB AVE	DEPT/JOB MIN	AVE	MIN
MARK	56,757	50,500	257PSB	55,860	50,500	55,500	55,500
						62,500	62,500
						50,500	50,500
						52,000	52,000
						58,800	58,800

```
*****DEPARTMENT MARK / JOB 257PSB SUBFOOT*****
NOT WITHIN: AVE=      $58,800.00  MIN=      $58,800.00
  WITHIN: AVE=      $55,860.00  MIN=      $50,500.00
*****
```

```
                257PTB  59,000    55,500    62,500    62,500
                                55,500    55,500
```

```
*****DEPARTMENT MARK / JOB 257PTB SUBFOOT*****
NOT WITHIN: AVE=      $55,500.00  MIN=      $55,500.00
  WITHIN: AVE=      $59,000.00  MIN=      $55,500.00
*****
```

```
*****DEPARTMENT MARK SUBFOOT*****
NOT WITHIN: AVE=      $55,500.00  MIN=      $55,500.00
  WITHIN: AVE=      $56,757.14  MIN=      $50,500.00
*****
```

Example: Creating a Cover Page for Each Sort Group in a Report

The following request prints the subheading "SUM OF PRICES AND QUANTITIES FOR THE region REGION" and then starts a new page containing the data for that region. The PAGE-BREAK command starts a new page after printing this data, prior to the subheading for the subsequent region:

```
TABLE FILE CENTORD
SUM LINEPRICE AS ''
QUANTITY AS ''
BY REGION NOPRINT PAGE-BREAK
BY STATE AS ''
ON REGION SUBHEAD NEWPAGE
"SUM OF PRICES AND QUANTITIES FOR THE <REGION REGION"
END
```

The first few pages of output follow:

PAGE 1

SUM OF PRICES AND QUANTITIES FOR THE EAST REGION

PAGE 2

CT	\$16,238,158.37	65,979
DC	\$70,928,546.26	274,714
DE	\$2,500,849.39	10,226
MA	\$34,010,314.29	131,956
MD	\$24,978,362.10	94,827
NH	\$4,985,236.56	20,752
NJ	\$38,906,712.15	154,974
NY	\$41,667,939.52	171,742
PA	\$27,830,850.54	104,456
RI	\$821,994.05	3,250
VT	\$2,751,969.47	10,631

PAGE 3

SUM OF PRICES AND QUANTITIES FOR THE NORTH REGION

PAGE 4

IA	\$2,469,227.24	10,068
IL	\$34,444,984.60	134,351
IN	\$12,477,236.78	50,124
KS	\$2,136,103.34	7,870
MI	\$47,979,137.95	191,671
MN	\$28,162,612.99	114,687
NA	\$1,027,220.04	3,040
OH	\$25,681,832.51	102,089
ON	\$12,699,111.42	49,142
WI	\$11,283,071.47	44,157

Note that without the PAGE-BREAK command, the subheading for each new region prints at the bottom of the page for the prior region's data.

Using Data in Headings and Footings

How to:

Insert Data in Headings and Footings

Reference:

Usage Notes for 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 9-463).
- ❑ Fields in headings and footings are evaluated as if they were 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 and Footing

This request displays the DEPARTMENT field in the heading and footing:

```
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
```

The output is:

```
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
```

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
<code><n or <n></code>	<code><50</code>	The next character starts in column 50.
<code><+n or <+n></code>	<code><+4</code>	The next character starts four columns from the last non-blank character.
<code><-n or <-n></code>	<code><-1</code>	The next character starts one column to the left of the last character and suppresses or writes over all or part of a field.
<code></n or </n></code>	<code></2</code>	Skip two lines.
<code><0X or <0X></code>	<code><0X</code>	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 skips one line more 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 starts 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"
```

Extending Heading and Footing Code to Multiple Lines

How to:

Extend Heading and Footing Code to Multiple Lines

A single line heading or footing code, between double quotation marks, can be a maximum of 32K characters. However, in some editors the maximum length of a line of code in a procedure is 80 characters. In cases like this, you can use the <0X spot marker to continue your heading onto the next line. The heading or footing content and spacing appears exactly as if typed on a single line.

Even if you do not need to extend your code beyond the 80-character line limit, this technique is convenient, since shorter lines may be easier to read on screen and to print on printers.

Procedure: How to Extend Heading and Footing Code to Multiple Lines

To extend the length of a single-line heading or footing beyond 80 characters:

1. Begin the heading or footing with double quotation marks.

2. Split the heading or footing content into multiple lines of up to 76 characters each, using the <OX spot marker at any point up to the 76th character to continue your heading onto the next line. (The four remaining spaces are required for the spot marker itself, and a blank space preceding it.)
3. Place the closing double quotation marks at the end of the final line of heading or footing code.

Example: Extending Heading and Footing Code to Multiple Lines

This request creates a sort heading coded on two lines. The <OX spot marker positions the first character on the continuation line immediately to the right of the last character on the previous line. No spaces are inserted between the spot marker and the start of a continuation line.

```
SET PAGE-NUM = OFF
JOIN STORE_CODE IN CENTCOMP TO STORE_CODE IN CENTORD
TABLE FILE CENTCOMP
HEADING
"Century Corporation Orders Report"
PRINT PROD_NUM QUANTITY LINEPRICE
BY STORE_CODE NOPRINT
BY ORDER_NUM
ON STORE_CODE SUBHEAD
"Century Corporation orders for store <STORENAME <OX
(store # <STORE_CODE|) <OX in <STATE|."
END
```

The partial output is:

Order Number:	Product Number#:	Quantity:	Line Total
Century Corporation orders for store Audio Expert (store # 1003CA) in CA.			
48108	1006	90	\$29,310.78
	1008	90	\$13,368.96
	1020	90	\$25,033.89
	1032	290	\$20,481.38
	1034	290	\$114,618.37
Century Corporation orders for store Audio Expert (store # 1003CO) in CO.			
54095	1006	12	\$3,645.42
	1008	12	\$1,926.35
	1020	12	\$3,314.28
	1032	211	\$15,983.61
	1034	211	\$87,868.51

Tip: Although it is demonstrated here for a sort heading, you can use this technique with any heading or footing line.

Example: Using Data in a Heading

The following example lists the employee's name, department, job description, and skill category in the heading:

```
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
```

The output is:

```
EMPLOYEE NAME           MARY SMITH
CURRENT DEPARTMENT      MIS
JOB TITLE               FILE QUALITY
*****
SKILL CATEGORY         FIQU
*****
```

Example: Using Direct Operators in Headings and Footings

You can use any prefix operator in a heading or footing to perform specific operations. This example prints the maximum, minimum, average, and total units sold:

```
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
```

The output is:

```
PAGE      1

MOST UNITS SOLD WERE      80
LEAST UNITS SOLD WERE    12
AVERAGE UNITS SOLD WERE  35
TOTAL UNITS SOLD WERE    645
```

This request prints the COUNTRY field, count of models, and average retail cost in a subfoot each time the country changes:

```
TABLE FILE CAR
BY COUNTRY NOPRINT SUBFOOT
"NUMBER OF MODELS IN COUNTRY <COUNTRY = <CNT.MODEL <0X
WITH AVERAGE COST OF <AVE.RCOST "
END
```

The output is:

```
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
```

This request prints the totals of units sold, returns, and actual sales:

```
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
```

The following special operators are specifically for use in 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 are 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 [Creating a Free-Form Report](#) on page 1009.

Creating Paging and Numbering

In this section:

Specifying a Page Break: PAGE-BREAK

Inserting Page Numbers: TABPAGENO

Controlling Report Page Numbering

Suppressing Page Numbers: SET PAGE

Preventing an Undesirable Split: NOSPLIT

The appearance of your report can be enhanced by controlling paging and page numbering. You can:

- Specify a page break: PAGE-BREAK.
- Insert page numbers: TABPAGENO.
- Specify the number of the first page, and continue page numbering across multiple reports: FOCFIRSTPAGE and FOCNEXTPAGE.
- Suppress page numbers: SET PAGE.
- Prevent an undesirable split: NOSPLIT.

Specifying a Page Break: PAGE-BREAK

How to:

Specify a Page Break

Reference:

Usage Notes for Page Breaks

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. Put the PAGE-BREAK command on the lowest-level sort field at which the page break is to occur. 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 [Conditionally Formatting Reports With the WHEN Clause](#) on page 411).

Reference: Usage Notes for Page Breaks

- ❑ Page headings and column titles appear at the top of each new page.
- ❑ 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 [Saving and Reusing Your Report Output](#) on page 421).
- ❑ When the request has a PAGE-BREAK, the GRANDTOTAL is on a page by itself.

Example: Specifying a Page Break

```
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.

Note: In a styled report, you can also create numbering of the form Page x of y using the TABLASTPAGE variable. For more information, see [Styling Reports](#) on page 491.

Example: Inserting Page Numbers

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.

Controlling Report Page Numbering**How to:**

Set the First Page Number for a Report

The SET FOCFIRSTPAGE command enables you to designate the first page number on a report. You can set FOCFIRSTPAGE to a specific number or the value of a Dialogue Manager variable. The &FOCNEXTPAGE variable enables you to establish consecutive page numbering across multiple reports.

When a report is processed, the variable &FOCNEXTPAGE is set to the number following the last page number in the report. This value can then be used as the first page number in a subsequent report, making the report output from multiple requests more useful and readable.

Consecutive page numbering can span multiple -INCLUDE commands.

If TABPAGENO is used in a request with FOCFIRSTPAGE, it correctly reflects the page number set by FOCFIRSTPAGE.

Syntax: How to Set the First Page Number for a Report

At the command line, in a FOCEXEC, or in a FOCUS-supported profile:

```
SET FOCFIRSTPAGE = {n|&var}
```

In a TABLE request:

```
ON TABLE SET FOCFIRSTPAGE {n|&var}
```

where:

n

Is the one- to six-digit number to be assigned to the first page of report output. The default value is 1.

&var

Is a Dialogue Manager variable whose value is used as the first page number of the report. &FOCNEXTPAGE is a system variable whose value is one greater than the last page of the prior report.

Example: Setting the Number of the First Page of a Report

This example runs two report requests, each of which uses TABPAGENO in its heading:

- ❑ The first report displays a list of movies.
- ❑ The second report displays movies with specific ratings. The SET FOCFIRSTPAGE command prior to the second report causes it to start with the next consecutive page number after the end of the first report.

The following procedure contains both report requests:

```

TABLE FILE MOVIES
HEADING
"MOVIES BY CATEGORY AND DIRECTOR:          PAGE <TABPAGENO "
" "
PRINT RATING TITLE
BY CATEGORY BY DIRECTOR
WHERE CATEGORY EQ 'ACTION' OR 'MUSICALS' OR 'COMEDY' OR 'CHILDREN'
WHERE DIRECTOR NE ' '
WHERE RATING NE 'NR'
END

-RUN
SET FOCFIRSTPAGE=&FOCNEXTPAGE

TABLE FILE MOVIES
HEADING
"MOVIES APPROPRIATE FOR CHILDREN:          PAGE <TABPAGENO "
" "
PRINT TITLE
BY CATEGORY BY DIRECTOR BY RATING
WHERE CATEGORY EQ 'ACTION' OR 'MUSICALS' OR 'COMEDY' OR 'CHILDREN'
WHERE DIRECTOR NE ' '
WHERE RATING EQ 'G' OR RATING CONTAINS 'PG'
END

```

The first report has pages 1 and 2. The output is:

MOVIES BY CATEGORY AND DIRECTOR:				PAGE	1
CATEGORY	DIRECTOR	RATING	TITLE		
-----	-----	-----	-----		
ACTION	MCDONALD P.	R	RAMBO III		
	SCOTT T.	PG	TOP GUN		
	SPIELBERG S.	PG	JAWS		
	VERHOVEN P.	R	ROBOCOP		
			R	TOTAL RECALL	
CHILDREN	BARTON C.	G	SHAGGY DOG, THE		
	DISNEY W.	G	BAMBI		
	GEROMINI	G	ALICE IN WONDERLA		
COMEDY	ABRAHAMS J.	PG	AIRPLANE		
	ALLEN W.	PG	ANNIE HALL		
	BROOKS J.L.	R	BROADCAST NEWS		
	HALLSTROM L.	PG13	MY LIFE AS A DOG		
	MARSHALL P.	PG	BIG		
	ZEMECKIS R.	PG	BACK TO THE FUTUR		
MUSICALS	ATTENBOROUGH R.	PG13	CHORUS LINE, A		
	FOSSE B.	PG	CABARET		

MOVIES BY CATEGORY AND DIRECTOR:				PAGE	2
CATEGORY	DIRECTOR	RATING	TITLE		
-----	-----	-----	-----		
MUSICALS	FOSSE B.	R	ALL THAT JAZZ		
	JEWISON N.	G	FIDDLER ON THE ROOF		

The second report starts on page 3. The output is:

MOVIES APPROPRIATE FOR CHILDREN:				PAGE	3
CATEGORY	DIRECTOR	RATING	TITLE		
-----	-----	-----	-----		
ACTION	SCOTT T.	PG	TOP GUN		
	SPIELBERG S.	PG	JAWS		
CHILDREN	BARTON C.	G	SHAGGY DOG, THE		
	DISNEY W.	G	BAMBI		
	GEROMINI	G	ALICE IN WONDERLAND		
COMEDY	ABRAHAMS J.	PG	AIRPLANE		
	ALLEN W.	PG	ANNIE HALL		
	HALLSTROM L.	PG13	MY LIFE AS A DOG		
	MARSHALL P.	PG	BIG		
	ZEMECKIS R.	PG	BACK TO THE FUTURE		
MUSICALS	ATTENBOROUGH R.	PG13	CHORUS LINE, A		
	FOSSE B.	PG	CABARET		
	JEWISON N.	G	FIDDLER ON THE ROOF		

Suppressing Page Numbers: SET PAGE

How to:

Suppress Page Numbers

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 382. 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 is specified, or the column titles, if there is no heading.

Preventing an Undesirable Split: NOSPLIT

How to:

Prevent an Undesirable Split

Reference:

Usage Notes for 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**

```
{ON|BY} fieldname NOSPLIT
```

where:

fieldname

Is the name of the sort field for which the sort groups are 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 is 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 [Saving and Reusing Your Report Output](#) on page 421).
- ❑ 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:

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

PAGE 2

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

Creating Paging and Numbering

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	
		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.

Suppressing Fields: SUP-PRINT or NOPRINT

How to:

Suppress Fields

Reference:

Usage Notes for Suppressing Fields

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

```
display fieldname {SUP-PRINT|NOPRINT}
{ON|BY} fieldname {SUP-PRINT|NOPRINT}
```

where:

display

Is any display command.

fieldname

Is a sort field or display field. The values of the field may be used, but they are not 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
```

results in a HOLD file that does not contain the DEPARTMENT field.

Example: Suppressing Fields

To print a list of employee names in alphabetical order, if you simply use the request

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

you get a report that lists the last names of employees in the order they were entered into the data source:

```
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:

```
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 output is:

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
UNIONDALE	E3	70	80
	B20	25	40
	C7	40	40

Also, consider the following example which does not display a COUNTRY column:

```
TABLE FILE CAR
SUM SALES BY COUNTRY
BY CAR
ON COUNTRY SUB-TOTAL SUP-PRINT PAGE-BREAK
END
```

The first part of the output is:

```
PAGE      1

CAR              SALES
---            -
JAGUAR          12000
JENSEN           0
TRIUMPH          0

*TOTAL ENGLAND
                12000
```

```
PAGE      2

CAR              SALES
---            -
PEUGEOT         0

*TOTAL FRANCE
                0
```

```
PAGE      3

CAR              SALES
---            -
ALFA ROMEO      30200
MASERATI         0

*TOTAL ITALY
                30200
```

Reducing a Report's Width: FOLD-LINE and OVER

In this section:

Compressing the Columns of Reports: FOLD-LINE

Decreasing the Width of a Report: 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

How to:

Compress Report Columns

Reference:

Usage Notes for Compressing Report Columns

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

```
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 is 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, as described in [Decreasing the Width of a Report: OVER](#) on page 396.
- ❑ 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 406.
- ❑ Up to 16 FOLD-LINE phrases can be used in a request.

Example: Compressing Report Columns

The following report places all fields following the DEPARTMENT field on another line:

```
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
```

The output is:

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

How to:
Decrease the Width of a Report

Reference:
Usage Notes for Decreasing Report Width

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.

Syntax: How to Decrease the Width of a Report

```
display fieldname1 OVER fieldname2 OVER fieldname3 ...
```

where:

```
display
```

Is any display command.

```
fieldname1, fieldname2, fieldname3
```

Are fields to be placed over each other, instead of printed beside each other in a row. The field names must be display fields.

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 [Using Data in Headings and Footings](#) on page 372.
- ❑ Text fields cannot be specified with OVER.

Example: Decreasing the Width of a Report

The following report stacks the display fields over each other:

```
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.

	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 looks like this:

DEPARTMENT MIS		PRODUCTION	
GROSS	\$50,499.12	GROSS	\$50,922.38
DED_AMT	\$28,187.25	DED_AMT	\$23,391.35
	NET \$22,311.88	NET	\$27,531.03

Positioning Columns: IN

How to:

Position Columns

Reference:

Usage Notes for Positioning Columns

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**

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 394), the positions apply to the line on which the referenced field occurs.

Example: Positioning Columns

The following request positions all of the report columns:

```
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.

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 places one extra space between the data columns in the matrix, and displays the ACROSS sets beginning in Position 30, as shown in the partial first page of the report below.

COUNTRY	CAR		
	ALFA ROMEO	AUDI	BMW
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 394), 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.

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

The following report request stacks the SALES field over the RETAIL_COST field:

```
TABLE FILE CAR
PRINT SALES IN 50 OVER RCOST IN 50
BY COUNTRY IN 10 BY MODEL
END
```

The output is:

COUNTRY	MODEL		
-----	-----		
ENGLAND	INTERCEPTOR III	SALES	0
		RETAIL_COST	17,850
	TR7	SALES	0
		RETAIL_COST	5,100
	V12XKE 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

Separating Sections of a Report: SKIP-LINE and UNDER-LINE

In this section:

Adding Blank Lines: SKIP-LINE

Underlining Values: 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

How to:

Add Blank Lines

Reference:

Usage Notes for Adding Blank Lines

Report information often stands out more clearly if there are lines 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 is 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.

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

The following example skips a line when the employee ID changes:

```
DEFINE FILE EMPLOYEE
INCREASE/D8.2M = .05*CURRE_SAL;
CURRE_SAL/D8.2M=CURRE_SAL;
NEWSAL/D8.2M=CURRE_SAL + INCREASE;
END

TABLE FILE EMPLOYEE
PRINT CURRE_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:

EMP_ID	LAST_NAME	FIRST_NAME		
-----	-----	-----		
071382660	STEVENS	ALFRED	CURRE_SAL	\$11,000.00
			INCREASE	\$550.00
			NEWSAL	\$11,550.00
112847612	SMITH	MARY	CURRE_SAL	\$13,200.00
			INCREASE	\$660.00
			NEWSAL	\$13,860.00
117593129	JONES	DIANE	CURRE_SAL	\$18,480.00
			INCREASE	\$924.00
			NEWSAL	\$19,404.00
119265415	SMITH	RICHARD	CURRE_SAL	\$9,500.00
			INCREASE	\$475.00
			NEWSAL	\$9,975.00
119329144	BANNING	JOHN	CURRE_SAL	\$29,700.00

Underlining Values: UNDER-LINE**How to:**

Underline Values

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

```
{ON|BY} fieldname UNDER-LINE [WHEN expression;
```

where:

fieldname

Is used so that a line is drawn when the value of the field changes. 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 382).

Example: Underlining Values

The following example adds an underline when the bank name changes:

```
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:

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

Controlling Column Spacing: SET SPACES

How to:

Control Column Spacing

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

```
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.

COUNTRY	CAR		BMW	
	ALFA ROMEO DEALER_COST	RETAIL_COST	DEALER_COST	RETAIL_COST
ITALY	16,235	19,565	.	.
W GERMANY	.	.	49,500	58,762

Creating New Column Titles: AS

How to:

Create Column Titles

Reference:

Usage Notes for New Column Titles

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 394.)

- ❑ 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 394.

- ❑ 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.
- ❑ The width allotted for column titles has no limit other than the memory available. It is initially set to 6K, but if that is not enough, the space is dynamically extended to accommodate the column title space required.

Example: Creating New Column Titles

The following example assigns new column titles for the LAST_NAME, FIRST_NAME, and EMP_ID fields:

```
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:

DEPARTMENT	EMPLOYEE NUMBER	NAME	
-----	-----	----	
MIS	112847612	MARY	SMITH
	117593129	DIANE	JONES
	219984371	JOHN	MCCOY
	326179357	ROSEMARIE	BLACKWOOD
	543729165	MARY	GREENSPAN
	818692173	BARBARA	CROSS
PRODUCTION	071382660	ALFRED	STEVENS
	119265415	RICHARD	SMITH
	119329144	JOHN	BANNING
	123764317	JOAN	IRVING
	126724188	ANTHONY	ROMANS
	451123478	ROGER	MCKNIGHT

Customizing Column Names: SET QUALTITLES

How to:

Customize Column Headings

Reference:

Usage Notes for Qualified Column Titles

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 qualified field names, see [Displaying Report Data](#) on page 45.)

Syntax: **How to Customize Column Headings**

```
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.

Column Title Justification

How to:

Justify Column Titles

Reference:

Usage Notes for Justifying Column Titles

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. 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 is 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
```

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

Customizing Reports With SET Parameters

How to:
Use SET Parameters in Requests

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

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:

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

Conditionally Formatting Reports With the WHEN Clause

How to:

Create Conditional Formatting

Reference:

Usage Notes for Conditional Formatting

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 [Conditionally Formatting Reports With the WHEN Clause](#) on page 411).

expression

Is any Boolean expression that is valid on the right side of a COMPUTE expression (see [Using Expressions](#) on page 323).

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:

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

You can print a report that selectively displays a SUBTOTAL and a SUBFOOT with two WHEN phrases:

```
TABLE FILE SALES
PRINT UNIT_SOLD RETAIL_PRICE
BY PROD_CODE
ON PROD_CODE SUBTOTAL
WHEN PROD_CODE CONTAINS 'B'
SUBFOOT
"PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP"
WHEN PROD_CODE CONTAINS 'C'
END
```

The relevant parts of the output are:

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
B17	29	\$1.89
	20	\$1.89
*TOTAL B17	49	\$3.78
.		
.		
.		
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 is 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 is displayed. If PROD_CODE contains the literal D, the subhead DISCONTINUED PRODUCT is 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:

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, a subtotal is calculated for each PROD_CODE, but the subfoot is displayed only when PROD_CODE contains the literal B:

```
SET PAGE-NUM = OFF
SET SCREEN = PAPER
-RUN
TABLE FILE SALES
PRINT UNIT_SOLD RETAIL_PRICE
BY PROD_CODE
ON PROD_CODE SUBTOTAL AND SUBFOOT
"PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP"
" "
WHEN PROD_CODE CONTAINS 'B'
END
```

The partial output is:

PROD_CODE	UNIT_SOLD	RETAIL_PRICE
-----	-----	-----
B10	60	\$.95
	30	\$.85
	13	\$.99
*TOTAL B10		
	103	\$2.79
PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP		
B12	40	\$1.29
	29	\$1.49
*TOTAL B12		
	69	\$2.78
PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP		
B17	29	\$1.89
	20	\$1.89
*TOTAL B17		
	49	\$3.78
PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP		
B20	15	\$1.99
	25	\$2.09
*TOTAL B20		
	40	\$4.08
PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP		
C13	25	\$1.99
*TOTAL C13		
	25	\$1.99
C17	12	\$2.09
*TOTAL C17		
	12	\$2.09

Example: Using Aggregation in the WHEN Clause

The following request prints a subfoot depending on the sum of the UNIT_SOLD field. Each subfoot displays the subtotal of the UNIT_SOLD field within its sort group:

```
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
```

The output is:

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

How to:

Control Empty Reports

Reference:

Usage Notes for Displaying Empty Reports

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

The EMPTYREPORT=ANSI setting introduces ANSI-compliant handling of reports with zero records, producing a one-line report containing only the specified missing data character. If the request is a COUNT operation, however, a zero is displayed for the missing values.

Syntax: **How to Control Empty Reports**

Use this command:

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

Valid values are:

ANSI

Produces a single-line report and displays the missing data character or a zero if a COUNT is requested. In each case, &RECORDS will be 0, and &LINES will be 1.

If the SQL Translator is invoked, ANSI automatically replaces OFF as the default setting for EMPTYREPORT.

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 or styled output formats.
- ❑ SET EMPTYREPORT = ON behaves as described regardless of ONLINE or OFFLINE settings.

10 | Saving and Reusing Your Report Output

When you run a report request, by default the data values are collected and presented in a viewable form, complete with column titles and formatting features. Instead of viewing the data values, you can save them to a special data file to:

- ❑ Display as a Web page, as a printed document, or in 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.
- ❑ Extract a data source to a structured extract file that retains information about the segment relationships in order to facilitate migration of data sources and reports between operating environments.

Topics:

- ❑ Saving Your Report Output
- ❑ Creating a HOLD File
- ❑ Holding Report Output in FOCUS Format
- ❑ Controlling Attributes in HOLD Master Files
- ❑ Keyed Retrieval From HOLD Files
- ❑ Using DBMS Temporary Tables as HOLD Files
- ❑ Creating SAVE and SAVB Files
- ❑ Creating a PCHOLD File
- ❑ Choosing Output File Formats
- ❑ Using Text Fields in Output Files
- ❑ Creating a Delimited Sequential File
- ❑ Saving Report Output in INTERNAL Format
- ❑ Creating a Structured HOLD File

Saving Your Report Output

In this section:

Naming and Storing Report Output Files

The following commands extract and save report output in a variety of file formats:

- ❑ **HOLD.** The HOLD command creates a data source containing the output of a report request. By default, the data is stored in binary format, but you can specify a different format, such as FOCUS, HTML, or Excel. 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. See [Creating a HOLD File](#) on page 423.
- ❑ **SAVE and SAVB.** The SAVE command is identical to a HOLD command, except that it does not create a Master File, and ALPHA is the default format. If you wish to create a SAVE file in BINARY format, use a variation of the SAVE command called SAVB.

As with a HOLD file, you can specify a variety of formats suitable for use with other software products. See [Creating SAVE and SAVB Files](#) on page 449.
- ❑ **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. See [Creating a PCHOLD File](#) on page 453.

Tip: When saving or holding output files, it is recommended to have an Allocation in place for the file. This is not applicable for PCHOLD files.

Naming and Storing Report Output Files

During a session, a report output file remains usable until it is erased or overwritten. A subsequent output file created during the same session replaces the initial version, unless you give it another name by using the AS phrase.

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.

By default, report output files created with HOLD, SAVE, or SAVB are written to temporary space. When the session ends, these files are no longer available unless you save the output to a specific location.

To save report output files to a specific location, use a FILEDEF or ALLOCATE command. 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 z/OS, you can dynamically allocate an output file using the DYNAM ALLOCATE or TSO ALLOCATE command.

See the *Overview and Operating Environments* manual for details.

Creating a HOLD File

How to:

- Create a HOLD File
- Create HOLD Files From Hot Screen
- Set the Default HOLD Format
- Query a HOLD Master File

You can use the HOLD command 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. Your report request can create a new data source, complete with a corresponding Master File, 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 specified in the Master File that are retrieved if the report is 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. See [Controlling Field Names in a HOLD Master File](#) on page 435.

- ❑ The HOLD command enables you to specify the appropriate formats for displaying or processing report output files in other software applications. See [Choosing Output File Formats](#) on page 455.
- ❑ 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.

Note: For information on writing programs to create HOLD files, see [Writing User-Coded Programs to Create HOLD Files](#) on page 1183.

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**

From a report request, use

```
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, use

```
HOLD [AS filename] [FORMAT fmt] [MISSING {ON|OFF}][VIA program]
```

where:

HOLD

Extracts and saves report output. BINARY is the default format used when the HOLD command is issued without an explicit format. The output is saved with an associated Master File.

Note: Change the default output format to ALPHA by issuing the SET HOLDFORMAT command.

hold_field

Is the name of the last display field in the request.

AS filename

Specifies a name for the HOLD file. If you do not specify a file name, HOLD becomes the default. Since each subsequent HOLD command overwrites the previous HOLD file, it is advisable 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. BINARY is the default format.

- To display as a Web page, choose: **HTML, HTMTABLE, DHTML**
- To display as a printed document, choose: **PDF, PS**
- To use in a text document, choose: **ALPHA, DOC, WP**
- To use in a spreadsheet application, choose: **DIF, EXCEL, EXL97, EXL2K [PIVOT], LOTUS, SYLK, TAB, TABT**
- To use in a database application, choose: **COMMA, COM, COMT, FOCUS, DB2, FUSION, SQL, SQLORA, SQLDBC**
- To use with a 3-GL program, choose: **INTERNAL**
- To use for additional reporting, choose: **ALPHA, BINARY, FOCUS**

- ❑ To use as a transaction file for modifying a data source, choose: [ALPHA](#), [BINARY](#)

For details about all available formats, see [Choosing Output File Formats](#) on page 455.

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 attribute. If the HOLD command specifies MISSING OFF, fields with the MISSING attribute are not carried over. For related information, see [Handling Records With Missing Field Values](#) on page 807. Also see the *Developing Applications* manual for the SET HOLDMISS, SET NULL, and SET HNODATA parameters, which control how missing values are propagated to alphanumeric and comma-delimited files.

VIA program

Calls a user-coded program to create the extract file. BINARY is the default format; 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

From the FOCUS command line or from Hot Screen, use the following after the report is executed

```
HOLD [AS filename][FORMAT fmt][MISSING {ON|OFF}][VIAprogram]
```

where:

fmt

Can be ALPHA, BINARY, DIF, SYLK, WP, INTERNAL, or FOCUS. If you omit the format, the default HOLD format (BINARY) is used.

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 when you issue the HOLD 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 Set the Default HOLD Format**

```
SET HOLDFORMAT = {BINARY|ALPHA}
```

or

```
ON TABLE SET HOLDFORMAT {BINARY|ALPHA}
```

where:

BINARY

 Sets the default HOLD file format to BINARY.

ALPHA

 Sets the default HOLD file format to ALPHA.

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 appears:

```
NUMBER OF RECORDS IN TABLE= 12 LINES= 12
```

You then see the message:

```
HOLDING...
```

To display the report generated by this request, either issue a report request against the HOLD file or 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.

Syntax: **How to Query a HOLD Master File**

If the HOLD format option you select creates a Master File, you can issue the following command to display the fields, aliases, and formats in the HOLD Master File:

```
? HOLD
```

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
```

Tip: You must issue the ? HOLD query in the same session in which the HOLD file is created.

Example: Reporting Against a 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 any text 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 contains the query command ? HOLD, which displays the fields, aliases, and formats in the associated Master File 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

? HOLD
```

The output is:

```
NUMBER OF RECORDS IN TABLE=      12      LINES=      12

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

How to:

Create HOLD Files in FOCUS Format

Reference:

Operating System Notes for HOLD Files in FOCUS Format

Controlling the FOCUS File Structure

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 from 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, use

```
ON TABLE HOLD [AS filename] FORMAT FOCUS [INDEX field1 field2 ...]
```

After a report request is run, use

```
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 becomes the default. Since each subsequent HOLD command overwrites the previous HOLD file, it is advisable 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.

Note: If you use a name longer than eight characters on OS/390, an eight-character member name is generated as described in the *Describing Data* manual. To relate the long name to the short member name, the \$ VIRT attribute is generated on the top line in the Master File. The resulting HOLD file is 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 are 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 a new data source and Master File are created on the disk that has WRITE permission and the most available space.

In z/OS, 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 done so. Since HOLD files are usually deleted, this is the desired default. However, if you want to save the Master File, allocate it 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 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 newly 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 are indexed; that is, FIELDTYPE=I is specified in the Master File. Up to four fields may be indexed.
- ❑ If the data in the HOLD file is longer than a page (4K for FOCUS data sources or 16K for XFOCUS data sources), it cannot be stored in a single segment. Data that is too long to become a single segment will become a parent segment with unique child segments. For a FOCUS data source, the fields will be grouped into normal FOCUS page size segments and added as unique segments up to the total maximum of 32K of data. For an XFOCUS data source, the root segment can hold the first 16K of data, and additional data up to the 32K total, will be placed in a single unique segment. BY fields must all occur in the portion of the data assigned to the root segment.

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 File 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 434.

Note: In environments that support the MODIFY facility, if the DIRECTHOLD parameter is set to ON, a Master File for the HOLD file and a sequential data source called FOC\$HOLD are created when the command HOLD FORMAT FOCUS is executed. The data in FOC\$HOLD is then loaded into the HOLD file using an internally generated MODIFY procedure. By default, this setting is OFF and the HOLD file is created directly without using a sequential data source and internal MODIFY procedure.

Example: Creating a HOLD File in FOCUS Format

The following example creates a subset of the CAR data source.

```
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.

The X1 Master File is created by the request:

```
FILE=X1, SUFFIX=FOC
SEGMENT=SEG01, SEGTYPE=S03
FIELDNAME=COUNTRY, ALIAS=E01, USAGE=A10, $
FIELDNAME=CAR, ALIAS=E02, USAGE=A16, $
FIELDNAME=MODEL, ALIAS=E03, USAGE=A24, $
FIELDNAME=SALES, ALIAS=E04, USAGE=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 because of the three BY fields and the FOCLIST FIELD.

```
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
SEGMENT=SEG01, SEGTYPE=S04
FIELDNAME=COUNTRY, ALIAS=E01, USAGE=A10, $
FIELDNAME=CAR, ALIAS=E02, USAGE=A16, $
FIELDNAME=MODEL, ALIAS=E03, USAGE=A24, FIELDTYPE=I, $
FIELDNAME=FOCLIST, ALIAS=E04, USAGE=I5, $
FIELDNAME=SALES, ALIAS=E05, USAGE=I6, $
```

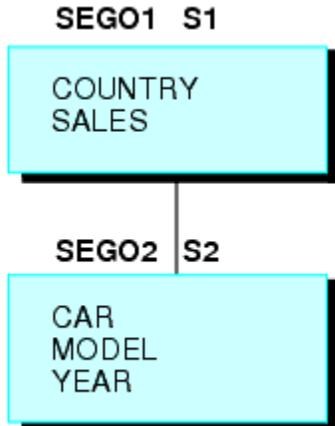
Example: Creating a Two-Segment FOCUS Data Source

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 is redundant.

```
TABLE FILE CAR
SUM SALES BY COUNTRY
SUM SALES BY COUNTRY BY CAR BY MODEL
ON TABLE HOLD AS X3 FORMAT FOCUS
END
```

This creates a two-segment FOCUS data source:



The Master File for this newly-created FOCUS data source is:

```
FILE=X3, SUFFIX=FOC
SEGMENT=SEG01, SEGTYPE=S01
  FIELDNAME=COUNTRY      ,ALIAS=E01      ,USAGE=A10      ,$
  FIELDNAME=SALES        ,ALIAS=E02      ,USAGE=I6       ,$
SEGMENT=SEG02, SEGTYPE=S02,PARENT=SEG01
  FIELDNAME=CAR          ,ALIAS=E03      ,USAGE=A16      ,$
  FIELDNAME=MODEL       ,ALIAS=E04      ,USAGE=A24      ,$
  FIELDNAME=SALES       ,ALIAS=E05      ,USAGE=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 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
SEGMENT=SEG01, SEGTYPE =S02
FIELDNAME=COUNTRY ,ALIAS=E01 ,USAGE=A10 ,FIELDTYPE=I,$
FIELDNAME=CAR ,ALIAS=E02 ,USAGE=A16 ,,$
FIELDNAME=SALES ,ALIAS=E03 ,USAGE=I6 ,,$
SEGMENT=SEG02, SEGTYPE =S01 ,PARENT=SEG01
FIELDNAME=MODEL ,ALIAS=E04 ,USAGE=A24 ,FIELDTYPE=I,$
FIELDNAME=SALES ,ALIAS=E05 ,USAGE=I6 ,,$
SEGMENT=SEG03, SEGTYPE =S02 ,PARENT=SEG02
FIELDNAME=BODYTYPE ,ALIAS=E06 ,USAGE=A12 ,,$
FIELDNAME=FOCLIST ,ALIAS=E07 ,USAGE=I5 ,,$
FIELDNAME=SALES ,ALIAS=E08 ,USAGE=I6 ,,$
```

Controlling Attributes in HOLD Master Files

In this section:

- Controlling Field Names in a HOLD Master File
- Controlling Fields in a HOLD Master File
- Controlling the TITLE and ACCEPT Attributes in the HOLD Master File

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 you change them.

- The SET ASNAMES command designates text specified in an AS phrase as the field name in the HOLD Master File, and concatenates it to the beginning of the first field name specified in an ACROSS phrase. See *Controlling Field Names in a HOLD Master File* on page 435.

- ❑ The SET HOLDLIST command restricts fields in HOLD 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. See [Controlling Fields in a HOLD Master File](#) on page 439.
- ❑ The SET HOLDATTR command propagates TITLE and ACCEPT attributes used in the original Master File to the HOLD Master File. See [Controlling the TITLE and ACCEPT Attributes in the HOLD Master File](#) on page 441.

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

How to:

Control Field Names in a HOLD Master File

Reference:

Usage Notes for Controlling Field Names in HOLD Files

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, and does not 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. FOCUS 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 is not used unless SET HOLDATTR was previously issued.
- ❑ To ensure that fields referenced more than once in a request have unique 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. Use single quotation marks around the non-standard field names when referring to them in the newly created Master File.
- ❑ 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.
- ❑ 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 does not work.

If an AS phrase is used for the fields in the ACROSS phrase, each new column has 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
SEGMENT=HOLD1, SEGTYPE=S01,$
  FIELDNAME=NATION      ,ALIAS=E01   ,USAGE=A10  ,ACTUAL=A12   ,$
  FIELDNAME=AUTOMOBILE ,ALIAS=E02   ,USAGE=A16  ,ACTUAL=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, if SET ASNAMES had 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
SEGMENT=HOLD2, SEGTYPE=S01,$
  FIELDNAME=CAR          ,ALIAS=E01   ,USAGE=A16  ,ACTUAL=A16   ,$
  FIELDNAME=SALES       ,ALIAS=E02   ,USAGE=I6   ,ACTUAL=I04   ,$
  FIELDNAME=AVERAGESALES,ALIAS=E03   ,USAGE=I6   ,ACTUAL=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
SEGMENT=HOLD3, SEGTYPE=S01,$
FIELDNAME=CASHENGLAND ,ALIAS=E01 ,USAGE=I6 ,ACTUAL=I04 ,,$
FIELDNAME=CASHFRANCE ,ALIAS=E02 ,USAGE=I6 ,ACTUAL=I04 ,,$
FIELDNAME=CASHITALY ,ALIAS=E03 ,USAGE=I6 ,ACTUAL=I04 ,,$
FIELDNAME=CASHJAPAN ,ALIAS=E04 ,USAGE=I6 ,ACTUAL=I04 ,,$
FIELDNAME=CASHW GERMANY ,ALIAS=E05 ,USAGE=I6 ,ACTUAL=I04 ,,$
```

Without the SET ASNAMES command, every field in the HOLD FILE is 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
SEGMENT=HOLD4, SEGTYPE=S0,$
FIELDNAME=ENGLAND ,ALIAS=E01 ,USAGE=I6 ,ACTUAL=I04 ,,$
FIELDNAME=FRANCE ,ALIAS=E02 ,USAGE=I6 ,ACTUAL=I04 ,,$
FIELDNAME=ITALY ,ALIAS=E03 ,USAGE=I6 ,ACTUAL=I04 ,,$
FIELDNAME=JAPAN ,ALIAS=E04 ,USAGE=I6 ,ACTUAL=I04 ,,$
FIELDNAME=W GERMANY ,ALIAS=E05 ,USAGE=I6 ,ACTUAL=I04 ,,$
```

Controlling Fields in a HOLD Master File

How to:

Control Fields in a HOLD 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 | ALLKEYS}
```

where:

PRINTONLY

Specifies that only those fields that would appear in the report are included in the generated HOLD file. Non-displaying fields in a request (those designated as NOPRINT fields) are not included in the HOLD file.

ALL

Specifies that all display fields referenced in a request appear in a HOLD file, including calculated values. ALL is the default value. OLD may be used as a synonym for ALL.

Note: Vertical sort (BY) fields specified in the request with the NOPRINT option are not included in the HOLD file even if HOLDLIST=ALL.

ALLKEYS

Propagates all fields, including NOPRINTed BY fields.

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 output is:

```
NUMBER OF RECORDS IN TABLE=      18          LINE=          18

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 TABLE request produces a HOLD file containing only fields that would appear 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:

NUMBER OF RECORDS IN TABLE=	18	LINES=	18
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

How to:

Control TITLE and ACCEPT Attributes

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 is 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 is 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. FOCUS is the default value.

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
```

produces this HOLD Master File:

```
FILE=HOLD5, SUFFIX=FOC
SEGMENT=SEG01, SEGTYPE=S02
  FIELDNAME=COUNTRY ,USAGE=E01 ,ACTUAL=A10
    TITLE='COUNTRY OF ORIGIN',
    ACCEPT=CANADA ENGLAND FRANCE ITALY JAPAN 'W GERMANY', $
  FIELDNAME=FOCLIST ,USAGE=E02 ,ACTUAL=I5 , $
  FIELDNAME=CAR ,USAGE=E03 ,ACTUAL=A16 ,
    TITLE='NAME OF CAR' , $
```

Keyed Retrieval From HOLD Files

How to:

Control Keyed Retrieval for a HOLD File

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, by using the SEGTYPE parameter in the Master File to identify which fields comprise the logical key for sequential files. When FIXRETRIEVE is:

- ❑ ON, the retrieval process stops when an equality or range test on the key holds true.
- ❑ 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 enables 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 enables 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 SEQ_NO, 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.

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, records that are out of sequence are not 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 is omitted, as retrieval stops as soon as a key value of 2 is encountered.

Using DBMS Temporary Tables as HOLD Files

In this section:

Column Names in the HOLD File

Primary Keys and Indexes in the HOLD File

How to:

Save Report Output as a Native Temporary Table Using Commands

Reference:

Temporary Table Properties for SAME_DB Persistence Values

You can create a report output file (that is, a HOLD file), as a native DBMS temporary table. This increases performance by keeping the entire reporting operation on the DBMS server, instead of downloading data to your computer and then back to the DBMS server.

For example, if you temporarily store report output for immediate use by another procedure, storing it as a temporary table instead of creating a standard HOLD file avoids the overhead of transmitting the interim data to your computer.

The temporary table columns are created from the following report elements

- Display columns
- Sort (BY) columns
- COMPUTE columns

except for those for which NOPRINT is specified.

The temporary table that you create from your report will be the same data source type (that is, the same DBMS) as the data source from which you reported. If the data source from which you reported contains multiple tables, all must be of the same data source type and reside on the same instance of the DBMS server.

You can choose between several types of table persistence.

You can create extract files as native DBMS tables with the following adapters:

- DB2 (on z/OS, UNIX, and Windows)
- Informix
- Microsoft SQL Server
- MySQL
- Oracle
- Teradata

Syntax: **How to Save Report Output as a Native Temporary Table Using Commands**

The syntax to save report output as a native DBMS temporary table is

```
ON TABLE HOLD [AS filename] FORMAT SAME_DB [PERSISTENCE persistValue]
```

where:

filename

Specifies the name of the HOLD file. If you omit AS *filename*, the name of the temporary table defaults to "HOLD".

Because each subsequent HOLD command overwrites the previous HOLD file, it is recommended to specify a name in each request to direct the extracted data to a separate file, thereby preventing an earlier file from being overwritten by a later one.

PERSISTENCE

Specifies the type of table persistence and related table properties. This is optional for DBMSs that support volatile tables, and required otherwise. For information about support for volatile tables for a particular DBMS, see [Temporary Table Properties for SAME_DB Persistence Values](#) on page 448, and consult your DBMS vendor documentation.

persistValue

Is one of the following:

VOLATILE

Specifies that the table is local to the DBMS session. A temporary synonym (a Master File and Access File), is generated automatically. It expires when the server session ends.

This is the default persistence setting for all DBMSs that support volatile tables.

For information about support for the volatile setting, and about persistence and other table properties, for a particular DBMS, see [Temporary Table Properties for SAME_DB Persistence Values](#) on page 448, and consult your DBMS vendor documentation.

GLOBAL_TEMPORARY

Specifies that while the table exists, its definition will be visible to other database sessions and users though its data will not be. A permanent synonym (a Master File and Access File), is generated automatically.

For information about support for the global temporary setting, and about persistence and other table properties, for a particular DBMS, see [Temporary Table Properties for SAME_DB Persistence Values](#) on page 448, and consult your DBMS vendor documentation.

PERMANENT

Specifies that a regular permanent table will be created. A permanent synonym (a Master File and Access File), is generated automatically.

Reference: Temporary Table Properties for SAME_DB Persistence Values

The following chart provides additional detail about persistence and other properties of temporary tables of different data source types that are supported for use with HOLD format SAME_DB.

DBMS	VOLATILE	GLOBAL_TEMPORARY
DB2	DB2 on UNIX, Windows, and DB2 for z/OS: a volatile table is created using the DECLARE GLOBAL TEMPORARY TABLE command with the ON COMMIT PRESERVE ROWS option. Declared global temporary tables persist and are visible only within the current session (connection). SESSION is the schema name for all declared global temporary tables.	DB2 Release 7.1 and up for z/OS only: a global temporary table is created using the CREATE GLOBAL TEMPORARY TABLE command. The definition of a created global temporary table is visible to other sessions, but the data is not. The data is deleted at the end of each transaction (COMMIT or ROLLBACK command). The table definition persists after the session ends.
Oracle	This type of table is not supported by Oracle.	The table's definition is visible to all sessions; its data is visible only to the session that inserts data into it. The table's definition persists for the same period as the definition of a regular table.
Teradata	A volatile table definition and data are visible only within the session that created the table and inserted the data. The volatile table is created with the ON COMMIT PRESERVE ROWS option.	A global temporary table persists for the same duration as a permanent table. The definition is visible to all sessions, but the data is visible only to the session that inserted the data. The global temporary table is created with the ON COMMIT PRESERVE ROWS option.

Column Names in the HOLD File

Each HOLD file column is assigned its name:

1. From the AS name specified for the column in the report request.
2. If there is no AS name specified, the name is assigned from the alias specified in the synonym. (The alias is identical to the column name in the original relational table.)

3. In all other cases, the name is assigned from the field name as it is specified in the synonym.

Primary Keys and Indexes in the HOLD File

A primary key or an index is created for the HOLD table. The key or index definition is generated from the sort (BY) keys of the TABLE command, except for undisplayed sort keys (that is, sort keys for which NOPRINT is specified). To determine whether a primary key or an index will be created:

1. If these sort keys provide uniqueness and do not allow nulls (that is, if in the synonym, the MISSING attribute column is unselected or OFF), and if the DBMS supports primary keys on the type of table being created, a primary key is created.
2. If these sort keys provide uniqueness but either
 - a. some of the columns allow nulls.
 - b. the DBMS does not support primary keys on the type of table being created then a unique index is created.
3. If these sort keys do not provide uniqueness, a non-unique index is created.
4. If there are no displayed sort keys (that is, no sort keys for which NOPRINT has not been specified), no primary key or index is created.

Creating SAVE and SAVB Files

How to:

Create a SAVE File

Create a SAVB File

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 455.

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**

From a report request, use

```
ON TABLE SAVE [AS filename] [FORMAT fmt] [MISSING {ON|OFF}]
```

or

```
save_field SAVE [AS filename] [FORMAT fmt] [MISSING {ON|OFF}]
```

After a report is executed, use

```
SAVE [AS filename] [FORMAT fmt] [MISSING {ON|OFF}]
```

where:

save_field

Is the name of the last field in the request, excluding BY or ACROSS fields.

AS filename

Specifies a name for the SAVE file. If you do not specify a file name, SAVE is used as the default. Since each subsequent SAVE command overwrites the previous SAVE file, it is advisable 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. ALPHA is the default format.

- To display as or in a Web page:

HTML, HTMTABLE, DHTML

- To use in a text document:

ALPHA, DOC, PDF, WP

- To use in a spreadsheet application:

DIF, EXCEL, EXL2K, LOTUS, SYLK

- To use in a database application:

COMMA, COM, COMT

For details about all available formats, see [Choosing Output File Formats](#) on page 455.

MISSING

Ensures that fields with the MISSING attribute set to ON are carried over into the SAVE file. MISSING OFF is the default attribute. See [Handling Records With Missing Field Values](#) on page 807.

Example: Creating a SAVE File

The following request extracts data from the EMPLOYEE data source and creates 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 appears after the records are retrieved.

The output is:

```

NUMBER OF RECORDS IN TABLE=      12  LINES=      12

ALPHANUMERIC RECORD NAMED  SAVE
FIELDNAME                   ALIAS           FORMAT          LENGTH

DEPARTMENT                  DPT             A10             10
LAST_NAME                   LN              A15             15
FIRST_NAME                   FN              A10             10

TOTAL                        TOTAL           TOTAL           35
```

Syntax: **How to Create a SAVB File**

From a request, use

```
ON TABLE SAVB [AS filename] [MISSING {ON|OFF}]
```

or

```
save_field SAVB [AS filename] [MISSING {ON|OFF}]
```

After a report is executed, use

```
SAVB [AS filename] [MISSING {ON|OFF}]
```

where:

save_field

Is the name of the last field in the request, excluding BY and ACROSS fields.

AS filename

Specifies a name for the SAVB file. If you do not specify a file name, SAVB is used as the default. Since each subsequent SAVB command overwrites the previous SAVB file, it is advisable 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 are carried over into the SAVB file. The default is MISSING OFF. See [Handling Records With Missing Field Values](#) on page 807.

Example: **Creating a SAVB File**

The following request extracts data from the SALES data source and creates 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 appears 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

```

Creating a PCHOLD File

How to:

Create a PCHOLD File

The PCHOLD command enables you to extract data from a Reporting server and return the output to a FOCUS client. See the *Overview and Operating Environments* manual for information about using FOCUS as a client to a Reporting server.

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 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. The output is saved with a Master File. For details about the behavior of PCHOLD, see [Creating a HOLD File](#) on page 423.

AS filename

Specifies a name for the PCHOLD file. If you do not specify a file name, PCHOLD becomes the default. Since each subsequent PCHOLD command overwrites the previous PCHOLD file, it is advisable 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. ALPHA is the default format.

- To display as or in a Web page, choose:

HTML, HTMTABLE, DHTML

- To display as a printed document, choose:

PDF, PS

- To use in a text document, choose:

ALPHA, DOC, WP

- To use in a spreadsheet application, choose:

DIF, EXCEL, EXL2K [PIVOT], LOTUS

- To use for additional reporting, choose:

ALPHA, BINARY

For details about all available formats, see [Choosing Output File Formats](#) on page 455.

Choosing Output File Formats

Reference:

FORMAT ALPHA	FORMAT INGRES
FORMAT BINARY	FORMAT INTERNAL
FORMAT COMMA	FORMAT LOTUS
FORMAT COM	FORMAT PDF
FORMAT COMT	FORMAT PDF OPEN/CLOSE
FORMAT DB2	FORMAT POSTSCRIPT (PS)
FORMAT DFIX	FORMAT PPT
FORMAT DHTML	FORMAT REDBRICK
FORMAT DIF	FORMAT SQL
FORMAT DOC	FORMAT SQLDBC
FORMAT EXCEL	FORMAT SQLINP
FORMAT EXL2K	FORMAT SQLMSS
FORMAT EXL2K FORMULA	FORMAT SQLODBC
FORMAT EXL2K PIVOT	FORMAT SQLORA
FORMAT EXL97	FORMAT SQLSYB
FORMAT FOCUS	FORMAT SYLK
FORMAT HTML	FORMAT TAB
FORMAT HTMTABLE	FORMAT TABT
FORMAT INGRES	FORMAT WP
	FORMAT XFOCUS

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 reference topics for the following formats.

ALPHA	DFIX	HTML	POSTSCRIPT (PS)	SQLORA
BINARY	DHTML	HTMTABLE	PPT	SQLSYB
COMMA	DIF	INGRES	REDBRICK	SYLK
COM	DOC	INTERNAL	SQL	TAB
COMT	EXCEL	LOTUS	SQLDBC	TABT
DB2	EXL2K	PDF	SQLINP	WP
	EXL2K FORMULA		SQLMSS	XFOCUS
	EXL2K PIVOT		SQLODBC	
	EXL97			
	FOCUS			

Reference: FORMAT ALPHA

Description: Saves report output as fixed-format character data and can be created as a HOLD file.

ALPHA is the default SAVE format. The output file contains data only.

Text fields are supported in ALPHA-formatted files. See [Using Text Fields in Output Files](#) on page 472.

To control missing data characters that are propagated to fields with the MISSING=ON attribute, use the SET HNODATA command. For more information, see the *Developing Applications* manual.

Use: For display in a text document. For further reporting in FOCUS, WebFOCUS, or Developer Studio. As a transaction file for modifying a data source.

Supported with the commands: HOLD, PCHOLD, SAVE.

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: FORMAT BINARY

Description: Saves report data and stores numeric fields as binary numbers. When created as a HOLD file, also creates a Master File.

BINARY is the default format for HOLD files. When created in BINARY format:

- ❑ 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 four-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. Do so with HOLD FORMAT INTERNAL. See [Saving Report Output in INTERNAL Format](#) on page 478.)

The output file contains data only.

Use: For further reporting in FOCUS, WebFOCUS, or Developer Studio. As a transaction file for modifying a data source.

Supported with the commands: HOLD, PCHOLD, SAVE.

Available in: WebFOCUS, Developer Studio, FOCUS.

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 does not have the safety feature of the double quote added within a text field containing a double quote.

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|QUOTE) is not supported. A comma within a number is interpreted as two separate columns by a destination application such as Microsoft Access.

The output file contains data only.

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, Developer Studio, FOCUS.

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 character fields. A second double quote (") is inserted adjacent to the existing one. For example, if you input Joe "Smitty" Smith, the output is 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|QUOTE) is not supported. A comma within a number is interpreted as two separate columns by a destination application such as Microsoft Access.
- ❑ To create a variable-length comma- or tab-delimited HOLD file that differentiates between a missing value and a blank string or zero value, use the SET NULL=ON command. For more information, see the *Developing Applications* manual.

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, PCHOLD.

Available in: WebFOCUS, Developer Studio, FOCUS.

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 character fields. A second double quote (") is inserted adjacent to the existing one. For example, if you input Joe "Smitty" Smith, the output is 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|QUOTEP) is not supported. A comma within a number is interpreted as two separate columns by a destination application such as Microsoft Access.
- ❑ To create a variable-length comma- or tab-delimited HOLD file that differentiates between a missing value and a blank string or zero value, use the SET NULL=ON command. For more information, see the *Developing Applications* manual.

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, PCHOLD.

Available in: FOCUS, Developer Studio, WebFOCUS.

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, Developer Studio, FOCUS.

Reference: **FORMAT DFIX**

Description: Creates a delimited output file. You can specify the delimiter, whether alphanumeric fields should be enclosed within a special character such as a double quotation mark, and whether the file should be generated with a header record containing the field names.

For more information, see [Creating a Delimited Sequential File](#) on page 473.

Use: For importing data to Windows-based applications such as MS Access and Excel.

Supported with the command: HOLD, PCHOLD.

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: **FORMAT DHTML**

Description: Provides HTML output that has most of the features normally associated with output formatted for printing such as PDF or PostScript output. You can create an HTML file (.htm) or a Web Archive file (.mht). The type of output file produced is controlled by the value of the HTMLARCHIVE parameter. HTMLARCHIVE=ON creates a Web Archive file.

Some of the features supported by format DHTML are:

- ❑ **Absolute positioning.** DHTML precisely places text and images inside an HTML report, allowing you to use the same StyleSheet syntax to lay out HTML as you use for PDF or PS output.
- ❑ **On demand paging.** On demand paging is available with SET HTMLARCHIVE=OFF.
- ❑ **PDF StyleSheet features.** For example, the following features are supported: grids, background colors, OVER, bursting, coordinated compound reports.

Note:

- ❑ The font map file for DHTML reports is dhtml.fmp.
- ❑ Legacy compound reports are not supported.

Use: For display as a Web page.

Supported with the commands: HOLD, PCHOLD, SAVE .

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: **FORMAT DIF**

Description: Captures the entire report output, excluding headings, footings, subheads, and subfoots, and creates a character file that can be easily incorporated into most spreadsheet packages.

For example, running a TABLE request with HEADING/FOOTING and ON TABLE HOLD FORMAT DIF does not display the report output with headings and footings. As a workaround, use another format (such as HTML, PDF, or EXL2K).

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.

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: **FORMAT DOC**

Description: Captures the entire report output, including headings, footings, and subtotals, and creates a text file with layout and line breaks that can be easily incorporated into most word processing packages. DOC format uses a form-feed character to indicate page control information.

Note: A request that contains ON TABLE HOLD FORMAT DOC results in a blank first page in the report when displayed in Microsoft XP Office. To eliminate this, include SET PAGE=NOPAGE in your request.

Use: For display in a text document.

Supported with the commands: HOLD, PCHOLD, SAVE.

The PCHOLD variation transfers the data from a Web server to a browser.

Available in: WebFOCUS, Developer Studio, FOCUS.

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. If you wish to have the ACROSS column titles appear, use HOLD FORMAT EXL2K.

Text and varchar (AnV) fields are not supported with FORMAT EXCEL. To include them, use HOLD FORMAT EXL2K. To use FORMAT EXL2K, you must have Excel 2000 installed.

Leading zeros do not appear for FORMAT EXCEL.

Since only single-line (single-cell) column titles are supported in format EXCEL reports, any additional column title rows are treated as data. For example, if you have a report with a multi-line (multi-cell) column title and you sort the column, the second (and so on) column title rows are sorted with the data. To avoid this, only select the data instead of the entire column when you select sorting options in 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, Developer Studio, FOCUS.

In FOCUS, 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 EXL2K**

Description: Generates fully styled reports in Excel 2000 HTML format. You must have Excel 2000 installed to use this output format.

ACROSS column titles are supported for EXL2K output format.

For EXL2K output format, a report can include 65,536 rows and/or 256 columns. Rows and columns in excess of these limits are dropped from the report.

Use: For display or processing in a spreadsheet application.

Supported with the commands: HOLD, PCHOLD, SAVE.

For details, and for information about working with EXL2K files, see [Working With Styled Output Formats](#) on page 629.

Available in: WebFOCUS, Developer Studio.

Reference: **FORMAT EXL2K FORMULA**

Description: Specifies that the report will be displayed as an Excel 2000 spreadsheet, with FOCUS totals and other calculated values translated to active Excel formulas. For details, see [Working With Styled Output Formats](#) on page 629.

Use: For display or processing in a spreadsheet application.

Supported with the commands: HOLD, PCHOLD, SAVE.

Available in: WebFOCUS, FOCUS, Developer Studio.

Reference: **FORMAT EXL2K PIVOT**

Description: Generates fully styled reports in Excel 2000 HTML format, with added pivoting capabilities. Requires Excel 2000 on your PC.

Use: For display or processing in a spreadsheet application.

Supported with the commands: HOLD, PCHOLD, SAVE.

Available in: WebFOCUS, FOCUS, Developer Studio.

Reference: **FORMAT EXL97**

Description: Enables you to view and save reports in Excel 97 that include full styling. For details on working with Excel formats, see [Working With Styled Output Formats](#) on page 629.

Leading zeros do not display for FORMAT EXL97.

Use: For display or processing in a spreadsheet application.

Supported with the command: HOLD, PCHOLD, SAVE.

Available in: FOCUS, WebFOCUS, Developer Studio.

Reference: **FORMAT FOCUS**

Description: Creates a FOCUS data source. Four files result: a HOLD data file, a HOLD Master File, and two work files. See [Holding Report Output in FOCUS Format](#) on page 429.

Text fields are supported for FOCUS output files. See [Using Text Fields in Output Files](#) on page 472.

Use: For further processing in a database application.

Supported with the command: HOLD.

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: **FORMAT HTML**

Description: Creates a complete HTML document that can be viewed in a Web browser. For more information, see Chapter 10, Styling Reports.

Use: For display as a Web page.

Supported with the commands: HOLD, PCHOLD, SAVE.

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: **FORMAT HTMTABLE**

Description: Creates an output file that contains only an HTML table. The output produced is not a complete HTML document.

Internal Cascading Style Sheets (CSS) are supported for FORMAT HTMTABLE. The CSS code is placed immediately before the TABLE command.

Use: For embedding reports and graphs in an existing HTML document.

Supported with the commands: HOLD, PCHOLD, SAVE.

Available in: WebFOCUS, Developer Studio, 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, Developer Studio, FOCUS when used as a client to the WebFOCUS Reporting Server.

Reference: **FORMAT INTERNAL**

Description: Saves report output without padding the values of integer and packed fields. See [Saving Report Output in INTERNAL Format](#) on page 478.

Use: For accurate processing by 3GL programs.

Supported with the command: HOLD, SAVB.

Available in: WebFOCUS, Developer Studio, FOCUS.

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 only).

Available in: WebFOCUS, Developer Studio, FOCUS.

In VM/CMS, the LOTUS file has a file type of PRN and allocates a scratch data set to the file HOLD.

Reference: **FORMAT PDF**

Description: Saves the report output in Adobe Portable Document Format (PDF), which enables precise placement of output (all formatting options such as headings, footings, and titles) correctly aligned on the physical page, so the report looks exactly as it does when printed.

If you have a wide PDF report, it is automatically paneled. However, the PANEL parameter has no effect on FORMAT PDF.

A PDF object is a page, hyperlink, or image. The Portable Document Format (PDF) limits the number of objects that a PDF document can contain. FOCUS imposes the following object limits for each PDF report:

Object	Limit
Pages	10,000
Images	900
Hyperlinks per page	500
Total pages with hyperlinks	100
Total hyperlinks	44,500

PDF format retains all formatting options, such as a headings, footings, and titles.

The following fonts are supported: Courier (fixed width), Times (proportional width), and Helvetica (proportional width). PDF format maps all fonts to Courier, Helvetica, or Times. The font styles that can be used are Normal (default), Bold, Italic, Underline, and combinations of these.

The following StyleSheet features are supported with PDF: PAGESIZE, ORIENTATION, UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, POSITION, SQUEEZE, HGRID, VGRID, BACKCOLOR. Note when you use BACKCOLOR with PDF reports, extra space is added to the top, bottom, right, and left of each cell of data in the report. This is for readability and to prevent data truncation.

Use: For display as a printed document. For information about compound reports and styling options, see [Working With Styled Output Formats](#) on page 629.

Supported with the commands: HOLD, PCHOLD, SAVE (WebFOCUS only).

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: **FORMAT PDF OPEN/CLOSE**

Description: Saves multiple reports into one PDF report.

Use: For combining multiple reports into a single PDF file, also known as a compound report.

Supported with the command: PCHOLD.

Available in: WebFOCUS, Developer Studio, 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.

PostScript format supports headings, footings, and totals. PS supports ISO Latin font encoding.

Use: For display as a printed document. For information about compound reports and styling options, see Chapter 10, Styling Reports.

Supported with the command: HOLD, PCHOLD.

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: **FORMAT PPT**

Description: Creates an output file in PowerPoint format in which each page of report output becomes a separate slide in the file with all styling applied.

Use: For use in a slide presentation.

Supported with the command: HOLD, SAVE.

Available in: WebFOCUS, Developer Studio, FOCUS.

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, Developer Studio, FOCUS when used as a client to the WebFOCUS Reporting Server.

Reference: FORMAT SQL

Description: Creates a DB2 for VM table, if you have the adapter for DB2 for VM and permission to create tables.

Use: For processing in a database application.

Supported with the command: HOLD.

Available in: WebFOCUS, Developer Studio, FOCUS.

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, Developer Studio, FOCUS.

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, Developer Studio, FOCUS when used as a client to the WebFOCUS Reporting Server.

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, Developer Studio, FOCUS when used as a client to the WebFOCUS Reporting Server.

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, Developer Studio, FOCUS when used as a client to the WebFOCUS Reporting Server.

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, Developer Studio, FOCUS.

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, Developer Studio, FOCUS when used as a client to the WebFOCUS Reporting Server.

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, SAVE.

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: FORMAT TAB

Description: Creates an output file in tab-delimited format. The TAB format includes a built-in safety feature, which allows embedded quotes within character fields. A second double quote (") is inserted adjacent to the existing one. For example, if you input Joe "Smitty" Smith, the output is Joe ""Smitty"" Smith. The TAB format also includes the following features:

- ❑ 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 HOLD command is used to create the output file. The Master File behaves exactly as in FORMAT ALPHA, except for the inclusion of double quotes.

Note: To create a variable-length comma- or tab-delimited HOLD file that differentiates between a missing value and a blank string or zero value, use the SET NULL=ON command. For more information, see the *Developing Applications* manual.

Use: For importing data to Windows-based applications such as MS Access and Excel.

Supported with the command: HOLD, SAVE, PCHOLD.

Available in: WebFOCUS, Developer Studio, FOCUS.

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 character fields. A second double quote (") is inserted adjacent to the existing one. For example, if you input Joe "Smitty" Smith, the output is 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 HOLD command is used to create the output file. 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.
- ❑ To create a variable-length comma- or tab-delimited HOLD file that differentiates between a missing value and a blank string or zero value, use the SET NULL=ON command. For more information, see the *Developing Applications* manual.

Use: For importing data to Windows-based applications such as MS Access and Excel.

Supported with the command: HOLD, SAVE, PCHOLD.

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: **FORMAT WP**

Description: Captures the entire report output, including headings, footings, and subtotals, and creates a text file that can easily be incorporated into most word processing packages.

Text fields are supported in WP format. See [Using Text Fields in Output Files](#) on page 472.

To control whether a carriage control character is included in column 1 of each page of the report output, use:

```
[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 z/OS, 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. In order to do so, issue one or a combination of the commands SET PRINT=OFFLINE, SET SCREEN=PAPER, or SET SCREEN=OFF.

In z/OS, 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 record length for HOLD FORMAT WP is 358 characters, 356 of which can represent data.

If you need the report width to remain fixed across releases for later processing of the output file, you can set the width you need using the SET WPMINWIDTH command. This parameter specifies the minimum width of the output file. It will be automatically increased if the width you set cannot accommodate the fields propagated to the output file in the request. On z/OS, The LRECL of the output file will be four bytes more than the report width because the file is variable length and needs an additional four bytes to hold the actual length of each record instance. In other operating environments, the length of the record is the value of WPMIDWIDTH.

FORMAT WP retains headings, footings, and subtotals.

Use: For display in a text document.

Supported with the commands: HOLD, PCHOLD, SAVE.

Available in: WebFOCUS, Developer Studio, FOCUS.

Reference: [FORMAT XFOCUS](#)

Description: Creates an XFOCUS data source.

Use: For further processing in a database application.

Supported with the command: HOLD.

Available in: WebFOCUS, Developer Studio, FOCUS.

Using Text Fields in Output Files

Reference:

Rules for Text Fields in Output Files

Text fields can be propagated to HOLD and SAVE files that have the following formats: ALPHA, WP, HTML, EXL2K, PDF, and FOCUS or XFOCUS. They can also be propagated to SAVB files. However, although a Master File is generated for format ALPHA, you cannot issue a TABLE request against a HOLD file of format ALPHA that contains text fields.

Reference: Rules for Text Fields in Output Files

- ❑ 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 indicates the end of the field.
- ❑ If there is no text for a field, a single period (.) followed by blanks appears 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.

See [Handling Records With Missing Field Values](#) on page 807.

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, conditional ©) in the FIXFORM format description, or some combination of the two.

Example: Applying Text Field Rules in HOLD Files

The following request extracts data to a HOLD file named CRSEHOLD:

```
TABLE FILE COURSES
PRINT COURSE_CODE DESCRIPTION
ON TABLE HOLD AS CRSEHOLD
END
```

This produces the following data in the HOLD file CRSEHOLD:

```
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 contains data for the first text field. After the end-of-text mark is written, the next text field appears. This formatting applies to all file formats except WP, in which the report is saved exactly as it appears on the screen.

Creating a Delimited Sequential File**How to:**

Create a Delimited Sequential File

Reference:

Usage Notes for HOLD FORMAT DFIX

You can use the HOLD FORMAT DFIX command to create an alphanumeric sequential file delimited by any character or combination of characters. You can also specify whether to enclose alphanumeric values in quotation marks or some other enclosure and whether to include a header record that lists the names of the fields.

A Master File and an Access File are created to describe the delimited sequential file that is generated. The SUFFIX value in the Master File is DFIX. The Access File specifies the delimiter, the enclosure character (if any), and whether there is a header record. The Master and Access Files are useful if you will later read the sequential file using WebFOCUS.

Syntax: **How to Create a Delimited Sequential File**

```
ON TABLE HOLD [AS filename] FORMAT DFIX
                DELIMITER delimiter [ENCLOSURE enclosure] [HEADER {YES|NO}]
```

where:

filename

Is the name of the file to be created. If you do not specify a name, the default name is HOLD.

delimiter

Consists of up to 30 printable or non-printable non-null characters. For a non-printable character, enter the hexadecimal value that represents the character. If you use a mixture of printable and non-printable characters, you must enter them all as hexadecimal values. To create a tab delimited file, you can specify the delimiter value as *TAB* or as its hexadecimal equivalent (0x09 on ASCII platforms or 0x05 on EBCDIC platforms).

Note that numeric digits and symbols used in numbers, such as a period (.), plus sign (+), or minus sign (-) cannot be used in the delimiter sequence.

enclosure

Consists of up to four printable characters used to enclose each alphanumeric value in the file. Most alphanumeric characters can be used as all or part of the enclosure sequence. However, numeric digits and symbols used in numbers, such as a period (.), plus sign (+), or minus sign (-) cannot be used in the enclosure sequence. Also note that, in order to specify a single quotation mark as the enclosure character, you must enter four consecutive single quotation marks. The most common enclosure is one double quotation mark.

HEADER {YES|NO}

Specifies whether to include a header record that contains the names of the fields in the delimited sequential file generated by the request.

Reference: **Usage Notes for HOLD FORMAT DFIX**

- ❑ Missing data is indicated by no data. So, with enclosure, a missing alphanumeric field is indicated by two enclosure characters, while a missing numeric field is indicated by two delimiters.
- ❑ Text fields are not supported with HOLD FORMAT DFIX.

- ❑ While HOLD FORMAT DFIX creates a single segment file, you can manually add segments to the resulting Master and Access File. In the Access File, you can specify a separate delimiter and/or enclosure for each segment.
- ❑ The FILETYPE of the generated sequential file on z/VM is FOCTEMP.

Example: Creating a Pipe-Delimited File

The following request against the CENTORD data source creates a sequential file named PIPE1 with fields separated by the pipe character (|). Alphanumeric values are not enclosed in quotation marks, and there is no header record:

```
TABLE FILE CENTORD
SUM QUANTITY LINEPRICE BY REGION BY YEAR
ON TABLE HOLD AS PIPE1 FORMAT DFIX DELIMITER |
END
```

The PIPE1 Master File specifies the SUFFIX value as DFIX:

```
FILENAME=PIPE1 , SUFFIX=DFIX , $
SEGMENT=PIPE1, SEGTYPE=S2, $
FIELDNAME=REGION, ALIAS=E01, USAGE=A5, ACTUAL=A05, $
FIELDNAME=YEAR, ALIAS=E02, USAGE=YY, ACTUAL=A04, $
FIELDNAME=QUANTITY, ALIAS=E03, USAGE=I8C, ACTUAL=A08, $
FIELDNAME=LINEPRICE, ALIAS=E04, USAGE=D12.2MC, ACTUAL=A12, $
```

The PIPE1 Access File specifies the delimiter:

```
SEGNAME=PIPE1, DELIMITER=|, HEADER=NO, $
```

The PIPE1 sequential file contains the following data. Each data value is separated from the next value by a pipe character:

```
EAST|2000|3907|1145655.77
EAST|2001|495922|127004359.88
EAST|2002|543678|137470917.05
NORTH|2001|337168|85750735.54
NORTH|2002|370031|92609802.80
SOUTH|2000|3141|852550.45
SOUTH|2001|393155|99822662.88
SOUTH|2002|431575|107858412.0
WEST|2001|155252|39167974.18
WEST|2002|170421|42339953.45
```

The following version of the HOLD command specifies both the delimiter and an enclosure character ("):

```
ON TABLE HOLD AS PIPE1 FORMAT DFIX DELIMITER | ENCLOSURE "
```

The Master File remains the same, but the Access File now specifies the enclosure character:

```
SEGNAME=PIPE1, DELIMITER=|, ENCLOSURE=", HEADER=NO, $
```

In the delimited file that is created, each data value is separated from the next by a pipe character, and alphanumeric values are enclosed within double quotation marks:

```
"EAST" | 2000 | 3907 | 1145655.77
"EAST" | 2001 | 495922 | 127004359.88
"EAST" | 2002 | 543678 | 137470917.05
"NORTH" | 2001 | 337168 | 85750735.54
"NORTH" | 2002 | 370031 | 92609802.80
"SOUTH" | 2000 | 3141 | 852550.45
"SOUTH" | 2001 | 393155 | 99822662.88
"SOUTH" | 2002 | 431575 | 107858412.01
"WEST" | 2001 | 155252 | 39167974.18
"WEST" | 2002 | 170421 | 42339953.45
```

This version of the HOLD command adds a header record to the generated file:

```
ON TABLE HOLD AS PIPE1 FORMAT DFIX DELIMITER | ENCLOSURE " HEADER YES
```

The Master File remains the same, but the Access File now specifies that the generated sequential file should contain a header record with column names as its first record:

```
SEGNAME=PIPE1, DELIMITER=|, ENCLOSURE=", HEADER=YES, $
```

In the delimited file that is created, each data value is separated from the next by a pipe character, and alphanumeric values are enclosed within double quotation marks. The first record contains the column names:

```
"REGION" | "YEAR" | "QUANTITY" | "LINEPRICE"
"EAST" | 2000 | 3907 | 1145655.77
"EAST" | 2001 | 495922 | 127004359.88
"EAST" | 2002 | 543678 | 137470917.05
"NORTH" | 2001 | 337168 | 85750735.54
"NORTH" | 2002 | 370031 | 92609802.80
"SOUTH" | 2000 | 3141 | 852550.45
"SOUTH" | 2001 | 393155 | 99822662.88
"SOUTH" | 2002 | 431575 | 107858412.01
"WEST" | 2001 | 155252 | 39167974.18
"WEST" | 2002 | 170421 | 42339953.45
```

Example: Creating a Tab-Delimited File

The following request against the CENTORD data source creates a sequential file named TAB1 with fields separated by a tab character:

```
TABLE FILE CENTORD
SUM QUANTITY LINEPRICE BY REGION BY YEAR
ON TABLE HOLD AS TAB1 FORMAT DFIX DELIMITER TAB
END
```

As the tab character is not printable, the TAB1 Access File specifies the delimiter using its hexadecimal value.

The following is the Access File in an EBCDIC environment:

```
SEGNAME=TAB1, DELIMITER=0x05, HEADER=NO, $
```

The following is the Access File in an ASCII environment:

```
SEGNAME=TAB1, DELIMITER=0x09, HEADER=NO, $
```

Example: Missing Data in the HOLD File

The following request against the CENTORD data source creates missing alphanumeric and numeric values in the resulting comma-delimited HOLD file:

```
DEFINE FILE CENTORD
AREA/A5 MISSING ON = IF REGION EQ 'EAST' THEN MISSING ELSE REGION;
MQUANTITY/I9 MISSING ON = IF REGION EQ 'WEST' THEN MISSING ELSE 200;
END
```

```
TABLE FILE CENTORD
SUM QUANTITY MQUANTITY LINEPRICE BY AREA BY YEAR
WHERE AREA NE 'NORTH' OR 'SOUTH'
ON TABLE HOLD AS MISS1 FORMAT DFIX DELIMITER , ENCLOSURE "
END
```

In the MISS1 HOLD file, the missing alphanumeric values are indicated by two enclosure characters in a row (“”) and the missing numeric values are indicated by two delimiters in a row (,,):

```
" ",2000,3907,600,1145655.77
" ",2001,495922,343000,127004359.88
" ",2002,543678,343000,137470917.05
"WEST",2001,155252,,39167974.18
"WEST",2002,170421,,42339953.45
```

Saving Report Output in INTERNAL Format

How to:

Suppress Field Padding in HOLD Files

Reference:

Usage Notes for Suppressing Padded Fields in HOLD Files

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, you must reformat the fields in the request in order to override the default ACTUAL formats that correspond to the USAGE formats in the Master File:

- ❑ Reformat the integer and packed fields that you do not want to be padded in the HOLD file to the correct display lengths.
- ❑ 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. See [Controlling Fields in a HOLD Master File](#) on page 439.

`fieldname/[In|Pn.d]`

Specify correct lengths in the formats for integer and packed fields where you wish to suppress padding. These formats override the ACTUAL formats used for the display formats in the Master File. See [Usage Notes for Suppressing Padded Fields in HOLD Files](#) on page 479.

Note that floating point double-precision (D) and floating point single-precision (F) 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 480.

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.
- ❑ 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 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.

- ❑ If you use the HOLDMISS=ON setting to propagate missing values to the HOLD file, short packed fields and fields with formats I1, I2, and I3 are not large enough to hold the missing value.

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. Alphanumeric fields also occupy full words.

```
TABLE FILE CAR
PRINT CAR COUNTRY RETAIL_COST DEALER_COST SEATS
ON TABLE HOLD AS DJG
END
```

The request creates the following Master File:

```
FILE=DJG, SUFFIX=FIX
SEGMENT=DJG, SEGTYPE=S0
FIELDNAME=CAR ,ALIAS=E01 ,USAGE=A16 ,ACTUAL=A16 , $
FIELDNAME=COUNTRY ,ALIAS=E02 ,USAGE=A10 ,ACTUAL=A12 , $
FIELDNAME=RETAIL_COST ,ALIAS=E03 ,USAGE=D7 ,ACTUAL=D08 , $
FIELDNAME=DEALER_COST ,ALIAS=E04 ,USAGE=D7 ,ACTUAL=D08 , $
FIELDNAME=SEATS ,ALIAS=E05 ,USAGE=I3 ,ACTUAL=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=PRINONLY
TABLE FILE CAR
PRINT CAR COUNTRY RETAIL_COST/I2 DEALER_COST/P3 SEATS/I1
ON TABLE HOLD AS HINT3 FORMAT INTERNAL
END
```

This creates the following Master File:

```
FILE=HINT3, SUFFIX=FIX
SEGMENT=HINT3, SEGTYPE=S0
FIELDNAME=CAR ,ALIAS=E01 ,USAGE=A16 ,ACTUAL=A16 , $
FIELDNAME=COUNTRY ,ALIAS=E02 ,USAGE=A10 ,ACTUAL=A10 , $
FIELDNAME=RETAIL_COST ,ALIAS=E03 ,USAGE=I6 ,ACTUAL=I02 , $
FIELDNAME=DEALER_COST ,ALIAS=E04 ,USAGE=P4 ,ACTUAL=P02 , $
FIELDNAME=SEATS ,ALIAS=E05 ,USAGE=I4 ,ACTUAL=I01 , $
```

The ACTUAL formats for the overridden fields are I2, P2, and I1. DEALER_COST has an ACTUAL of P2 because P3, the format override, means 3 display digits that can be stored in 2 actual digits. Note that the alphanumeric field is also not padded.

Creating a Structured HOLD File

How to:

- Activate Structured HOLD Files for a Request
- Create a Structured HOLD File
- Specify Options for Generating Structured HOLD Files

Reference:

- Elements Included in a Structured HOLD File
- Elements Not Included in a Structured HOLD File
- Structural and Behavioral Notes

Structured HOLD Files facilitate migration of data sources and reports between operating environments.

Other HOLD formats capture data from the original sources and may retain some implicit structural elements from the request itself. However, they do not propagate most of the information about the original data sources accessed and their inter-relationships to the HOLD Master File or data source. Structured HOLD files, however, extract the data to a structure that parallels the original data sources. Subsequent requests against the HOLD file can use these retained data relationships to recreate the same types of relationships in other environments or in other types of data sources.

A Structured HOLD File can be created in ALPHA, BINARY, or FOCUS format:

- ❑ A Structured HOLD file created in either ALPHA or BINARY format is a flat file that saves the segment instances that contain the data that satisfy the conditions of the TABLE request. Multiple segments are generated based on the original structure read by the TABLE request. Segments are identified by assigning a RECTYPE for differentiation. Child segments in the original data source become a unique segment in the HOLD file
- ❑ A Structured HOLD file in FOCUS format uses normal FOCUS segments to retain the original structure.

In all cases the HOLD file contains all of the original segment instances required to provide the complete report based on the TABLE request itself. Regardless of the display command used in the original request (PRINT, LIST, SUM, COUNT), the Structured HOLD File is created as if the request used PRINT. Aggregation is ignored.

The HOLD file contains either all of the fields in the structure identified by the request that are used to satisfy the request, or all of the display fields and BY fields. The file does not contain DEFINE fields not specifically referenced in the request. It does contain all fields needed to evaluate any DEFINE fields referenced in the request.

Structured HOLD files are only supported for TABLE and TABLEF commands. They can be created anywhere a HOLD file is supported. You must activate Structured HOLD files in a specific request by issuing the ON TABLE SET EXTRACT command in the request prior to creating the Structured HOLD File.

Syntax: **How to Activate Structured HOLD Files for a Request**

```
ON TABLE SET EXTRACT {ON|*|OFF}
```

where:

ON

Activates Structured HOLD Files for this request and extracts all fields mentioned in the request.

Activates Structured HOLD Files for this request and indicates that a block of extract options follows. For example, you can exclude specific fields from the Structured HOLD File. For information, see [How to Specify Options for Generating Structured HOLD Files](#) on page 483.

OFF

Deactivates Structured HOLD files for this request. OFF is the default value.

Syntax: **How to Create a Structured HOLD File**

Before issuing the HOLD command, activate Structured HOLD Files for the request by issuing the ON TABLE SET EXTRACT command described in [How to Activate Structured HOLD Files for a Request](#) on page 482. Then issue the HOLD command to create the Structured HOLD File:

```
[ON TABLE] {HOLD|PCHOLD} [AS name] FORMAT {ALPHA|BINARY|FOCUS}
```

where:

name

Is the name of the HOLD file. If omitted, the name becomes HOLD by default.

FORMAT

Is ALPHA, BINARY or FOCUS.

Note: You can issue a SET command to set the default HOLD format to either ALPHA or BINARY:

```
SET HOLDFORMAT=ALPHA  
SET HOLDFORMAT=BINARY
```

Syntax: How to Specify Options for Generating Structured HOLD Files

To specify options for creating the extract, such as excluding specific fields, use the * option of the SET EXTRACT command:

```
ON TABLE SET EXTRACT *
EXCLUDE = (fieldname1, fieldname2, fieldname3 , ..., fieldnamen), $
FIELDS={ALL|EXPLICIT}, $
ENDEXTRACT
ON TABLE HOLD AS name FORMAT {ALPHA|BINARY|FOCUS}
```

where:

```
EXCLUDE=(fieldname1, fieldname2, fieldname3, ..., fieldnamen)
```

Excludes the specified fields from the HOLD file.

```
, $
```

Is required syntax for delimiting elements in the extract block.

```
ALL
```

Includes all real fields and all DEFINE fields that are used in running the request.

```
EXPLICIT
```

Includes only those real fields and DEFINE fields that are in the display list or the BY sort field listing. DEFINE fields that are not explicitly referenced, and fields that are used to evaluate DEFINES, are not included.

```
ENDEXTRACT
```

Ends the extract block.

Example: Creating a Structured HOLD File in ALPHA Format

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
ON TABLE SET EXTRACT ON
ON TABLE HOLD FORMAT ALPHA
END
```

This request produces the following HOLD Master File:

```
FILENAME=HOLD      , SUFFIX=FIX      , $
  SEGMENT=EMPINFO, SEGTYPE=S0, $
    FIELDNAME=RECTYPE, ALIAS=R, USAGE=A3, ACTUAL=A3, $
    FIELDNAME=LAST_NAME, ALIAS='LN', USAGE=A15, ACTUAL=A15, $
    FIELDNAME=FIRST_NAME, ALIAS='FN', USAGE=A10, ACTUAL=A10, $
    FIELDNAME=DEPARTMENT, ALIAS='DPT', USAGE=A10, ACTUAL=A10, $
    FIELDNAME=ED_HRS, ALIAS='OJT', USAGE=F6.2, ACTUAL=A06, $
  SEGMENT=PAYINFO, SEGTYPE=S0, PARENT=EMPINFO, $
    FIELDNAME=RECTYPE, ALIAS=1, USAGE=A3, ACTUAL=A3, $
    FIELDNAME=SALARY, ALIAS='SAL', USAGE=D12.2M, ACTUAL=A12, $
    FIELDNAME=JOBCODE, ALIAS='JBC', USAGE=A3, ACTUAL=A03, $
```

Note the RECTYPE field generated for ALPHA or BINARY Structured HOLD files. Each record in the HOLD file begins with the RECTYPE to indicate the segment to which it belonged in the original structure. The root segment has RECTYPE=R. The RECTYPES for other segments are sequential numbers assigned in top to bottom, left to right order.

Following are the first several records in the HOLD file:

```
R  STEVENS      ALFRED      PRODUCTION 25.00
1      11000.00A07
1      10000.00A07
R  SMITH       MARY        MIS         36.00
1      13200.00B14
R  JONES      DIANE       MIS         50.00
1      18480.00B03
1      17750.00B02
R  SMITH      RICHARD     PRODUCTION 10.00
1      9500.00A01
1      9050.00B01
```

Example: Creating a Structured HOLD File in FOCUS Format

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
ON TABLE SET EXTRACT ON
ON TABLE HOLD FORMAT FOCUS
END
```

This request produces the following HOLD Master File:

```
FILENAME=HOLD      , SUFFIX=FOC      , $
  SEGMENT=EMPINFO, SEGTYPE=S0, $
    FIELDNAME=LAST_NAME, ALIAS='LN', USAGE=A15, $
    FIELDNAME=FIRST_NAME, ALIAS='FN', USAGE=A10, $
    FIELDNAME=DEPARTMENT, ALIAS='DPT', USAGE=A10, $
    FIELDNAME=ED_HRS, ALIAS='OJT', USAGE=F6.2, $
  SEGMENT=PAYINFO, SEGTYPE=S0, PARENT=EMPINFO, $
    FIELDNAME=SALARY, ALIAS='SAL', USAGE=D12.2M, $
    FIELDNAME=JOBCODE, ALIAS='JBC', USAGE=A3, $
```

Example: Reconstituting a Structured HOLD File

The following request reconstitutes the original FOCUS data source from the Structured HOLD File created in *Creating a Structured HOLD File in ALPHA Format* on page 483:

```
TABLE FILE HOLD
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
ON TABLE SET EXTRACT ON
ON TABLE HOLD AS RECONST FORMAT FOCUS
END
```

This request produces the following Master File:

```
FILENAME=RECONST   , SUFFIX=FOC     , $
  SEGMENT=EMPINFO, SEGTYPE=S0, $
    FIELDNAME=LAST_NAME, ALIAS='LN', USAGE=A15, $
    FIELDNAME=FIRST_NAME, ALIAS='FN', USAGE=A10, $
    FIELDNAME=DEPARTMENT, ALIAS='DPT', USAGE=A10, $
    FIELDNAME=ED_HRS, ALIAS='OJT', USAGE=F6.2, $
  SEGMENT=PAYINFO, SEGTYPE=S0, PARENT=EMPINFO, $
    FIELDNAME=SALARY, ALIAS='SAL', USAGE=D12.2M, $
    FIELDNAME=JOBCODE, ALIAS='JBC', USAGE=A3, $
```

The following request prints the report output:

```
TABLE FILE RECONST
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
END
```

The output is:

DEPARTMENT	SALARY	LAST_NAME	FIRST_NAME	JOBCODE	ED_HRS
MIS	\$27,062.00	CROSS	BARBARA	A17	45.00
	\$25,775.00	CROSS	BARBARA	A16	45.00
	\$21,780.00	BLACKWOOD	ROSEMARIE	B04	75.00
	\$18,480.00	JONES	DIANE	B03	50.00
		MCCOY	JOHN	B02	.00
	\$17,750.00	JONES	DIANE	B02	50.00
	\$13,200.00	SMITH	MARY	B14	36.00
	\$9,000.00	GREENSPAN	MARY	A07	25.00
	\$8,650.00	GREENSPAN	MARY	B01	25.00
	PRODUCTION	\$29,700.00	BANNING	JOHN	A17
\$26,862.00		IRVING	JOAN	A15	30.00
\$24,420.00		IRVING	JOAN	A14	30.00
\$21,120.00		ROMANS	ANTHONY	B04	5.00
\$16,100.00		MCKNIGHT	ROGER	B02	50.00
\$15,000.00		MCKNIGHT	ROGER	B02	50.00
\$11,000.00		STEVENS	ALFRED	A07	25.00
\$10,000.00		STEVENS	ALFRED	A07	25.00
\$9,500.00		SMITH	RICHARD	A01	10.00
\$9,050.00		SMITH	RICHARD	B01	10.00

Example: Excluding Fields From Structured HOLD Files

This request excludes the SALARY field used for sequencing.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
ON TABLE SET EXTRACT *
EXCLUDE=(SALARY), $
ENDEXTRACT
ON TABLE HOLD FORMAT FOCUS
END
```

This request produces the following HOLD Master File:

```
FILENAME=HOLD , SUFFIX=FOC , $
SEGMENT=EMPINFO, SEGTYPE=S0, $
FIELDNAME=LAST_NAME, ALIAS='LN', USAGE=A15, $
FIELDNAME=FIRST_NAME, ALIAS='FN', USAGE=A10, $
FIELDNAME=DEPARTMENT, ALIAS='DPT', USAGE=A10, $
FIELDNAME=ED_HRS, ALIAS='OJT', USAGE=F6.2, $
SEGMENT=PAYINFO, SEGTYPE=S0, PARENT=EMPINFO, $
FIELDNAME=JOBCODE, ALIAS='JBC', USAGE=A3, $
```

Reference: Elements Included in a Structured HOLD File

Structured HOLD files contain all original segment instances required to complete the TABLE or TABLEF request. Regardless of the display command used in the original request (PRINT, LIST, SUM, or COUNT), the structured HOLD file will be created as if the command was PRINT.

Specifically, the extract file contains the following elements:

- ❑ All real fields named in the request such as display objects, sort fields, and fields used in selection criteria (WHERE/IF tests).
Note that fields referenced multiple times in a request are included only once in the HOLD file.
- ❑ Fields used in FILTER FILE condition.
- ❑ Prefix operators are ignored except for ALL. (which just affects the amount of data collected and does not imply a calculation).
- ❑ Field based reformatting (FIELD1/FIELD2=) causes the original field and the format field to be included.
- ❑ A GROUP field if referenced explicitly or when all of its members are referenced in the request.

Note: If a group member is specifically excluded (EXCLUDE) or not referenced, its GROUP is not added to the extract Master File (this applies to nested and overlapping groups as well). If a GROUP and its elements are all named in a request, the GROUP is not added as a real field in the extract HOLD file.

- ❑ For FIELDS=ALL, all DEFINE fields used in the request become real fields in the structured HOLD File and are included along with other fields used in the DEFINE expression (including other DEFINE fields). Use EXCLUDE to reduce the number of fields included in the EXTRACT output.
- ❑ For FIELDS=EXPLICIT, display objects and sort fields are included. DEFINE fields become real fields if referenced in the request, but fields used to create them will not be included unless referenced explicitly. This reduces the number of fields returned in the request.

Reference: Elements Not Included in a Structured HOLD File

- ❑ Prefix operators on WHERE fields are evaluated in data selection but not included in the extract output.
- ❑ Prefix operators on display objects are ignored (except ALL).
- ❑ Using Structured HOLD File syntax in MATCH, MORE, and GRAPH requests produces error messages and exits the procedure.

- ❑ WHERE/IF TOTAL tests are not supported in Structured HOLD File requests and result in cancellation of the request.
- ❑ Reformatting of real fields is ignored (only the real field is included).
- ❑ Computed fields are not included, but fields used in COMPUTE expressions are included in the extract file.

Reference: Structural and Behavioral Notes

- ❑ Structured HOLD File requests are subject to the same limitations on number and size of fields as any other TABLE request.

Structural Notes

- ❑ The following SET parameters are turned off or ignored in Structured HOLD File requests:
 - ❑ AUTOINDEX
 - ❑ AUTOPATH
 - ❑ AUTOSTRATEGY
 - ❑ EXTHOLD
 - ❑ EXTSORT
 - ❑ HOLDATTR
- ❑ All SET and ON TABLE SET commands used to control output format are ignored in creating the extract file.
- ❑ Alternate views are respected and reflected in the structure of the extract file.
- ❑ Indexed views specified in the request are respected and reflected in the structure of the output file.
- ❑ If a request would generate a file containing two independent orphan segments because the parent segment is specifically excluded, a dummy system segment is created in the to act as parent of the two unrelated segments. There is only one instance of data for that segment. Both orphan segments refer to that system segment as parent. If the parent is missing because it was not mentioned in the request, it is activated during the request and included as the parent the segments.
- ❑ In the event that two unique (U) segments are included without the parent segment, the unique segments are converted to segments with SEGTYPE S0 that reference the system segment as parent.
- ❑ JOIN and JOIN WHERE structures are supported.

SQL Optimization Notes

- ❑ SQL optimization for aggregation must be turned off for EXTRACT requests.

BY/ACROSS/FOR Notes

- ❑ BY and ACROSS sort fields become additional display objects.
- ❑ BY. . .ROWS and ACROSS . . .COLUMNS function only as implicit WHEREs to limit field values included.
- ❑ FOR fields are included.
- ❑ RECAP fields are excluded (like COMPUTEs).
- ❑ Summarization fields referencing previously identified fields are ignored in creating Structured HOLD Files. These include: SUMMARIZE, RECOMPUTE, SUBTOTAL, SUB-TOTAL ACROSS-TOTAL, ROW-TOTAL and COLUMN-TOTAL.

Formatting Notes

- ❑ Structured HOLD File processing ignores all formatting elements, including: IN, OVER, NOPRINT, SUP-PRINT, FOLD-LINE, SKIP-LINE, UNDER-LINE, PAGE-BREAK, TABPAGENO, AS, and column title justification. However, fields referenced within formatting commands, such as HEADING, FOOTING, SUBHEAD, and SUBFOOT, and any WHEN expressions associated with them, are included.
- ❑ All STYLE and STYLESHEET commands are ignored in producing extract output.
- ❑ AnV and AnW fields are supported. TX fields are exported in FOCUS files only.
- ❑ In the event that the FIELDNAME and ALIAS are the same for a real field and that field is redefined as itself (possibly with a different format), two fields are created in the HOLD Master File with identical field names and aliases. In this situation, the second version of the field can never be accessed if referenced by name. You can use FIELDS=EXPLICIT to include only the second version of the field. The following DEFINE illustrates an example of creating a duplicate field name and alias:

```
DEFINE FILE CAR
COUNTRY/A25=COUNTRY;
END
```

DBA Notes

- ❑ DBA controls on source files are respected when running Structured HOLD File requests with the exception of RESTRICT=NOPRINT, where fields named in a request are not displayed (such fields cannot be exported and should be specifically EXCLUDED).
- ❑ DBA restrictions do not carry over to the HOLD Master File.

Reconstituting Extract Files

- ❑ To reconstitute a FOCUS or flat file from a Structured HOLD file, you use the same syntax used to generate the Structured HOLD File. The ON TABLE SET EXTRACT syntax must be used to preserve multipath structures.
- ❑ All reconstituted FOCUS segments are SEGTYPE=S0, as neither KEY nor INDEX information is retained. An INDEX can be reinserted using REBUILD INDEX.

11 Styling Reports

A styled report is a report produced as format HTML, PDF, PostScript, Excel 2000, or Excel 97 that uses different default and/or user-specified formats from a standard FOCUS report. You can specify various characteristics of your report and format report components individually using a StyleSheet, or you can let the report be created using the default styles associated with these output formats.

A StyleSheet enables you to format and produce attractive reports that highlight key information.

You can use a StyleSheet to:

- ❑ Format report components individually.
- ❑ Format data that meets specified conditions.
- ❑ Create macros that enable you to streamline your formatting specifications.

Some advanced features of styled reports work differently depending on which format you choose for your report output. For detailed information about working with each type of styled output format, see [Working With Styled Output Formats](#) on page 629.

Topics:

- ❑ Introduction to Styled Reports
- ❑ Choosing an Output Format
- ❑ Styling Reports With StyleSheets
- ❑ Creating a Styled Report
- ❑ Styling the Page Layout
- ❑ Specifying Font Format in a Report
- ❑ Identifying Report Components
- ❑ Reusing FOCUS StyleSheet Declarations With Macros
- ❑ FOCUS StyleSheet Attribute Inheritance
- ❑ Conditionally Formatting in a StyleSheet

Introduction to Styled Reports

In this section:

Choosing a Type of Style Sheet

A StyleSheet describes how you want your report to look. You can create a StyleSheet within a report request or as a separate file. Either way, it consists of a series of declarations. Each declaration identifies a report component whose characteristics you wish to define (such as a heading, column, or grand total) and describes the desired characteristics of that component.

For HTML reports, you can also use external Cascading Style Sheets and enable internal Cascading Style Sheets. For details, see [Cascading Style Sheets](#) on page 605.

Unless otherwise noted, all StyleSheet references in this document refer to FOCUS StyleSheets.

Style sheets enable you to create extremely detailed formats for every line, column, or value in your report. In most cases, you will want to use the formatting facilities judiciously to make important information stand out. FOCUS has its own StyleSheets and also supports Cascading Style Sheets for HTML reports, as described in [Cascading Style Sheets](#) on page 605.

For some types of formatting you can choose between using a style sheet or report syntax. Style sheets enable you to centralize and reuse formatting logic. This provides you with several advantages:

- ❑ **Productivity.** By using just a few lines of code—a single style sheet—you can format dozens of reports, reducing the development time of each report.
- ❑ **Easy maintenance.** You can change formatting for dozens of reports at one time by editing a single style sheet.
- ❑ **Consistent appearance.** Your enterprise can guarantee a consistent look for its reports by assigning the same style sheets to them.
- ❑ **Rapid reformatting.** You can change a report's appearance quickly and easily by switching the style sheet assigned to it.
- ❑ **Prioritizing.** You can focus on your first priority report content because you can quickly address report presentation by applying an existing style sheet.

You can apply several formatting techniques to save yourself time and effort. Most of these techniques enable you to use code provided for you by FOCUS, or to leverage code that you write yourself:

- ❑ **Inheritance and overrides.** Each report component inherits attributes from its "parent" report component. This powerful feature lets you define common formatting in a single declaration for a parent component, and lets descendant components automatically inherit the formatting, while enabling you to override the inherited values when you wish. By designing your StyleSheet to take advantage of inheritance, you can write less code and quickly update formatting for multiple report components.

For example, if you declare all the report's data to be blue, all data in all columns will be displayed as blue; if you also declare all vertical sort (BY) columns to be orange, this will override the blue for sort columns, which will be displayed as orange; and if you also declare the EMP_ID sort column green, this will override the orange and be displayed as green. For more information, see [FOCUS StyleSheet Attribute Inheritance](#) on page 581.

- ❑ **Macros.** If you are going to specify the same attribute and value in several declarations in a StyleSheet, you can create a macro that enables you to apply the attribute repeatedly throughout the StyleSheet without coding it each time. Then, if you need to change the value, you can change it once "in the macro" and have the change applied automatically throughout the StyleSheet.
- ❑ **Samples.** FOCUS comes with several sample StyleSheets, which you can apply to a report. You can also use a sample as a template, first copying it and then customizing the copy.
- ❑ **Defaults.** Many FOCUS StyleSheet attributes have default values. Instead of explicitly specifying every attribute, you can omit some and accept their defaults. For example, you can accept the default font instead of specifying a font. You can find each attribute's default value documented where its syntax is described.

For example, if there are several parts of a report that you wish to emphasize (such as titles of important columns, data values that exceed a threshold, and sort headings), and you want all of these to be bold and purple, you can define a macro that sets font style to bold and color to purple, and then apply the macro to all of these report components.

Example: Specifying Formatting for the Order Revenue Report

This report displays the order number, order date, and total order revenue for Century Corporation for the third quarter of 2001:



page 1

Date Of Order:	Order Number:	Order Total:
Oct 1, 01	59613	\$1,556.37
Oct 2, 01	58360	\$5,894.04
Oct 3, 01	59327	\$672,994.57
Oct 4, 01	58373	\$2,817.45
Oct 6, 01	54577	\$220,408.00
Oct 7, 01	68950	\$3,887.45
Oct 9, 01	48840	\$4,587.05
Oct 10, 01	48922	\$964,642.12
Oct 11, 01	48974	\$133,720.98

The report is formatted by a FOCUS StyleSheet and by formatting commands in the report procedure itself. The procedure is shown below, followed by the StyleSheet file, OrderRev:

```

TABLE FILE CENTORD
1. HEADING
1. " "
1. "Century Corporation"
1. " "
1. "Order Revenue - 2001 Q3"
1. " "
1. "page <TABPAGENO"
1. " "
2. SUM ORDER_DATE/MtDY ORDER_NUM LINEPRICE AS 'Order,Total:'
   BY LOWEST 9 ORDER_DATE NOPRINT
   WHERE (ORDER_DATE GE '2000/10/01') AND (ORDER_DATE LE '2000/12/31');
3. ON TABLE SET SQUEEZE ON
4. ON TABLE SET STYLE SHEET OrderRev
   ON TABLE HOLD FORMAT PDF
END
    
```

OrderRev Style Sheet

```

5. TYPE=Report, GRID=Off, UNITS=Inches, TOPGAP=0.06, BOTTOMGAP=0.06, $
6. TYPE=Data, FONT='Times', $
7. TYPE=Data, BACKCOLOR=Aqua, COLOR=Navy,
7.     WHEN=LinePrice GT 500000, $
7. TYPE=Data, COLUMN=LINEPRICE, BACKCOLOR=Aqua, COLOR=Navy, STYLE=Bold,
7.     WHEN=LinePrice GT 500000, $
8. TYPE=Title, FONT='Helvetica', $
9. TYPE=Heading, FONT='Helvetica', STYLE=Bold, SIZE=14, JUSTIFY=Center,
9.     BACKCOLOR=Dark Turquoise, COLOR=White, $
9. TYPE=Heading, LINE=6, BACKCOLOR=White, COLOR=Dark Turquoise, $
9. TYPE=Heading, LINE=7, BACKCOLOR=White, $

```

1. Adds a page heading to the report.
2. Reformats the order date from (for example) 2001/10/03 to Oct. 3, 01.
3. Aligns the heading with the report(tm)s margins instead of the page(tm)s margins.
4. Identifies a StyleSheet file to format the report.
5. Increases spacing between report lines.
6. Uses a proportional serif font for the report(tm)s data.
7. Highlights each order that totals more than \$500,000 by applying a navy font and an aqua background, and by bolding the order total.
8. Uses a proportional sans serif font for the report(tm)s column titles.
9. Formats the report(tm)s heading by centering it, applying a larger sans serif font, coloring most of it with a dark turquoise background and white lettering, and applying the inverse coloring to the page number (the sixth line of the heading).

This is only a summary of what these formatting instructions do; you can find complete explanations in the topics that describe each formatting feature.

The formatting logic that you apply to your own reports may be briefer or more extensive than this example, depending on the report and on what formatting you choose to apply.

Choosing a Type of Style Sheet

You can choose between two types of style sheets to format a report:

- ❑ **FOCUS StyleSheets** (often abbreviated to "StyleSheets"), the native FOCUS style sheet language. These provide you with the flexibility to format reports in many display formats, including HTML, PDF, Excel 2000, and PostScript. You can choose between saving the StyleSheet as a separate file, which you can assign to multiple reports, or saving it within one report request.

If you are generating a report in HTML format, you can boost its performance, and increase the number of formatting options available to it, by having the FOCUS StyleSheet dynamically generate an "internal" Cascading Style Sheet (CSS). (CSS is the standard style sheet language designed for HTML documents; the *internal* CSS generated by FOCUS is internal to the report output, instead of being saved as a separate file.)

- ❑ **External Cascading Style Sheets**, the standard style sheet language designed for HTML documents. You can apply an external Cascading Style Sheet to any FOCUS report in HTML format. (An *external* Cascading Style Sheet is one that is saved as a separate file, instead of within the document it formats, and so is "external" to the document.)

How do you choose between the two types of style sheets? Consider choosing:

- ❑ **A FOCUS StyleSheet** if you want to display a report in different display formats, such as PDF and Excel 2000. FOCUS StyleSheets support many kinds of display formats, but Cascading Style Sheets work for reports in HTML format only.
- ❑ **An external Cascading Style Sheet** for any of the following reasons:
 - ❑ Your enterprise already uses Cascading Style Sheets to format HTML documents, and it wants reports to conform to these same presentation guidelines.
 - ❑ You want to apply the same formatting to other kinds of HTML documents in your enterprise.

Choosing an Output Format

Reference:

Sample FOCUS StyleSheet Files

You can choose from several different formats when you create a styled report. Some formats are designed to be used with specific applications; in order to be used with those applications, they must be transferred to a PC using FTP. For example, you can choose:

- ❑ **HTML format.** You can display your report as an HTML page, which is optimized for display in a Web browser. HTML supports most style sheet options, especially when used with an internal Cascading Style Sheet. An HTML report opens in your Web browser.

Note: SET STYLMODE=FIXED turns off your browser(tm)s HTML formatting for that report. The resulting report appears in a fixed font without colors and other Web capabilities.

- ❑ **Print format-PDF** (the Adobe® Acrobat® Portable Document Format) and **PostScript (PS).**

You can display your report as a PDF document, which is useful when you want the report to look the same whether displayed on screen or in print. PDF (Adobe Acrobat's Portable Document Format) is most often used to distribute and share electronic documents using the web. It is especially useful if you want a report to maintain its presentation and layout regardless of a user's browser or printer type. PDF prints and displays a document consistently, regardless of the application software, hardware, and operating system used to create or display the document. The report opens in Adobe® Acrobat® or Acrobat Reader. To display a PDF report, a computer must have Adobe Acrobat Reader installed. For free downloads of Acrobat Reader, go to <http://www.adobe.com>.

Limit: Adobe Acrobat's PDF format limits the number of pages, hyperlinks, and images in a document. For information about what limits this creates for a FOCUS report in PDF format, see [Saving and Reusing Your Report Output](#) on page 421.

You can display your report as a PostScript (PS) document, which is a print-oriented page description language most often used to send a report directly to a printer. While used less frequently as an online display format, you can display PS report output on your monitor before printing it.

To display rather than print a PostScript report, a computer must have a third-party PostScript application installed, such as GSview (a graphical interface for Ghostscript).

If you are sending a PS report to a printer, you can select the size of the paper on which to print the output. The PostScript code that is generated works on PS printers that support Language Level 2 or above. It is ignored, without harmful effects, on Level 1 printers.

You can also select page orientation and choose among a wide range of paper sizes and combine several reports into one. PDF and PostScript reports require font metric files and font map files to be accessible to FOCUS. FOCUS comes with a set of basic fonts, and you can add other fonts as needed. For details on working with PDF and PostScript reports, see [Working With PostScript and PDF Reports](#) on page 687.

- **Spreadsheet formats.** You can display your report as an Excel 2000 or 97 spreadsheet, where you can work with the data in Excel. You can create Excel 2000 HTML-based format (with variations for Excel 2000 PivotTable and Excel 2000 FORMULA), or Excel 97 HTML-based format. Note that Excel 97 format produces the same report output as HTML format. It is useful if you want your Web browser to open the file in Excel rather than display it as an HTML page. These formats supply the following tools:
 - **Excel 2000 Worksheet (format EXL2K).** The Excel 2000 format supports most StyleSheet attributes, allowing for full report formatting. The computer on which the report is being displayed must have Microsoft Excel 2000 or higher installed.

In addition, FOCUS supports two Excel 2000 variations: EXL2K FORMULA and EXL2K PIVOT. When either of these formats is specified, additional processing is done. For information on using these options, see [Working With Excel 2000 and Excel 97 Reports](#) on page 635.

- ❑ **Excel 2000 FORMULA (EXL2K FORMULA).** If you display a report using the format variation EXL2K FORMULA, the resulting spreadsheet will contain Excel formulas that calculate and display the results of any type of summed information (such as column totals, row totals, and sub-totals).
- ❑ **Excel 2000 PivotTable (format EXL2K PIVOT).** Generates a fully functional Excel PivotTable, an Excel tool for analyzing complex data.
- ❑ **Excel 97 spreadsheet (format EXL97).** Excel 97 is an HTML-based display format that opens in Excel 97 or higher and supports report formatting. The computer on which the report is being displayed must have Microsoft Excel 97 or higher installed.

Reference: Sample FOCUS StyleSheet Files

Several sample StyleSheet files are shipped with FOCUS. You can apply any of these sample StyleSheets directly to a report, or you can use them as templates to create your own customized StyleSheets. If you use them as templates, be sure to make a copy of the original file and edit the copy, leaving the original unchanged.

Following are the sample StyleSheets available. On VM, they have FILETYPE FOCSTYLE. On z/OS, they are members in a data set allocated to ddname FOCSTYLE:

FOCUS StyleSheet	File/Member Name
Elegant Look	CLASSIC
Formal Corporate Presentation	CORPORAT
Gold and brown backgrounds for selected components	BOYSCOUT
Orange and yellow backgrounds for selected components	CITRUS
Bold headings	DEFLT
Gray backgrounds for selected components	DEFLT1
Dark pink and gray backgrounds for selected components	DEFLT2
Blue and gray backgrounds for selected components	DEFLT3
Blue and gray backgrounds for selected components	DEFLTBLU
Light pink backgrounds for selected components	ELEPHANT
Purple and fuchsia backgrounds for selected components	FUCHSIA

FOCUS StyleSheet	File/Member Name
Brown and pink backgrounds for selected components	MARBLE
Yellow and purple backgrounds for selected components	OLIVE
Dark gray and green backgrounds for selected components	PAPER
Dark blue and light gray backgrounds for selected components	SALES
Light yellow and blue backgrounds for selected components	SLATE
Sporty look with sports team colors such as orange and blue	SPORTS
Light green and gray backgrounds for selected components	TEAL
Red, White, and Blue	USA
Brown and yellow backgrounds for selected components	WOOD

Styling Reports With StyleSheets

In this section:

What Is a StyleSheet?

What Is a Style?

Comparison of Reports With and Without StyleSheets

Creating a StyleSheet

StyleSheet Syntax

Improving FOCUS StyleSheet Readability

Adding a Comment to a FOCUS StyleSheet

Checking StyleSheet Syntax

To use a StyleSheet, follow these 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: You can change some formatting features, such as the page parameters, with a SET command without creating a StyleSheet.

2. Activate the StyleSheet you have chosen, or create a StyleSheet in a report request.
3. Create a PostScript, HTML, PDF (Acrobat's Portable Document Format), Excel 2000, or Excel 97 file that contains the formatted report.

Use FTP to transfer this file to Windows, if necessary.

4. Print the report on a PostScript printer or open it in the appropriate application. This step is highly dependent on the equipment and software at your site. See your system administrator for instructions.

What Is a StyleSheet?

A StyleSheet is a group of declarations, in a text file or in a report request, that describe how you want your report to look. These declarations:

- ❑ 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.

FOCUS uses the same default style for all report components if you do not create a StyleSheet.

With a StyleSheet, you only have to define the styles of those components to be displayed differently from the default style. Any component not specifically formatted in your StyleSheet either uses the default style or inherits a style from a higher-level component. Inheritance is discussed in [FOCUS StyleSheet Attribute Inheritance](#) on page 581.

When you create a styled report, your page layout differs from a standard FOCUS report. This variation is a function of the page layout parameters FOCUS uses for the two kinds of reports:

- ❑ For unstyled reports, FOCUS uses the parameters LINES, PAPER, PANEL, and WIDTH to define the page layout.
- ❑ For styled reports, 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 with a SET command-inherits default values.

What Is a Style?

Reference:

Reproducing StyleSheet Examples

A style is a description of the physical characteristics of a report component; it consists of four basic 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.

You can also define such attributes as background colors, grids, borders, and images in a StyleSheet declaration.

Reference: [Reproducing StyleSheet Examples](#)

The sample reports use the default page size specification for PDF and PS reports, 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.

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), and the maximum number of characters in a report panel (PANEL). FOCUS then uses the typeface and size defined by your printer setup for all data in the report.

In a styled report, you can specify measurement units such as inches or points, and you can control column width or spacing. You can also change typefaces, type size, and type style for any part of the report.

In a PostScript or PDF report, you can also set margins on the top, bottom and sides of the report, and you can set the page size for letters, envelopes, and many other types of paper. The following table compares your options 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 italic.	FOCUS uses the default font specified in your printer setup for the entire report.
Colors	You can select a color for the text or background.	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 for PostScript and PDF reports.	You can specify the number of lines per page within the limits of your printer setup.
Page orientation	You can select either portrait or landscape for PostScript, PDF, and EXL2K reports.	FOCUS uses the default orientation specified in your printer setup.

	With StyleSheets	Without StyleSheets
Page margins	You can specify the top, bottom, left, and right margins, measured in inches, centimeters, or points for PostScript and PDF reports.	FOCUS uses the default page margins specified in your printer setup.
Justification	You can justify individual report components.	You can justify column titles.

Creating a StyleSheet

How to:

Create a StyleSheet in a Report Request

Activate an Existing StyleSheet File

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 [How to Create a StyleSheet in a Report Request](#) on page 504.
- ❑ 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 [How to Activate an Existing StyleSheet File](#) on page 505.

You can take advantage of most StyleSheet options without ever having to create a StyleSheet.

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.

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, in which the font for the entire report and the amount of space to the right of the EMP_ID field have been changed:

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

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 ignores the settings for LINES, PAPER, PANEL, and WIDTH. Each display format has its own set of defaults. For example, HTML defaults to a proportional font, while PDF defaults to a monospace font.

OFF uses default styles. In this case, FOCUS uses the settings for LINES, PAPER, PANEL, and WIDTH, and 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 [Creating a Styled Report](#) on page 508.

stysheet is the one- to 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 ignores the settings for LINES, PAPER, PANEL, and WIDTH.

StyleSheet Syntax

A StyleSheet consists of a series of declarations that describe how your report will 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 and a justification.
- ❑ 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 conditional styling.

In your StyleSheet, include declarations for only those components whose format to change. Within each declaration, include only those formatting attributes to change.

Each declaration in a StyleSheet 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 509 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.

Improving FOCUS StyleSheet Readability

There are many ways to structure your StyleSheet declarations in order to make the StyleSheet easy to read. You can do any one, or a combination, of the following:

- ❑ Begin a declaration in any column using blank spaces or tabs.
- ❑ Include blank lines between declarations.
- ❑ Create declarations in all uppercase, all lowercase, or mixed case.
- ❑ Use more than one declaration to format a single report component.
- ❑ Include blank spaces or tabs in between the attribute, equal sign (=), value, comma, and dollar sign (\$).
- ❑ Split a single declaration across a line. The declaration will continue to be processed until the dollar sign terminates it. For example, you can split a declaration like this:

```
TYPE=HEADING, FONT=ARIAL,
SIZE=14, STYLE=BOLD, $
```

- ❑ Split an *attribute=value* pair across a line. Use the backslash (\) character as continuation syntax at the end of the first line if you are splitting an attribute or value in a declaration across a line. For example:

```
TYPE=TITLE, COLUMN=N2, STY\
LE=BOLD+ITALIC, COLOR=BLUE, $
```

Adding a Comment to a FOCUS StyleSheet

You can add comments to a StyleSheet to give context to a declaration. Comments do not affect StyleSheet behavior.

You can add a comment:

- ❑ **On a declaration line.** Add the desired text after the dollar sign (\$). For example,
`TYPE=HEADING, STYLE=BOLD, COLOR=BLUE, SIZE=14, $ Sample comment`
- ❑ **On its own line.** Begin the line with either a dollar sign (\$), or a hyphen and an asterisk (-*), followed by the desired text. For example,
`-* This is a sample comment`
`$ This is another sample comment`

Note: You can add comments anywhere in your request, not only in StyleSheets.

Checking StyleSheet Syntax

How to:

Check 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 verify whether the specified fonts are available or whether the font names are spelled correctly.

Creating a Styled Report

How to:

Create a Styled Report

In order to create a report that was formatted using a StyleSheet, you must generate an output file containing the formatted report.

Syntax: How to Create a Styled Report

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 format [AS filename]
```

where:

filename

Assigns a 1- to 8-character file name or ddname to the report output. The default file name is HOLD.

format

Can be one of the following:

HTML for HTML output.

PDF for PDF output.

PS or POSTSCRIPT for PostScript output.

EXL2K for Excel 2000 output. You can also use EXL2K PIVOT for Excel 2000 output with a pivot table or EXL2K FORMULA for Excel 2000 output with formulas.

EXL97 for Excel 97 output.

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).

Styling the Page Layout

In this section:

Selecting Page Size, Orientation, and Color

Setting Page Margins

Displaying Current Settings: The ? SET STYLE Query

In a styled report, FOCUS uses page parameters to format the page layout. These parameters have default values that remain in effect unless you change them.

Selecting Page Size, Orientation, and Color

How to:

Set Page Size

Set Page Orientation

Control the Paper Source Used by a PostScript Printer

Set Page Color

Reference:

Page Size, Orientation, and Color Attributes

You can select the page size, page orientation (portrait or landscape), and page color for your report. The default page size is letter (8.5 x 11 inches), but you can select from many other sizes, including legal and envelopes.

In a PostScript report, you can use the SET PSPAGESETUP command to include PostScript code that automatically tells a PostScript printer to set its paper source to the size of paper specified by PAGESIZE.

Reference: [Page Size, Orientation, and Color Attributes](#)

Attribute	Description	Applies to
PAGESIZE	Sets page size.	PDF PS
ORIENTATION	Sets page orientation.	PDF PS EXL2K
PAGECOLOR	Sets page color.	HTML report with internal Cascading Style Sheet

Syntax: How to Set Page Size

This syntax applies to a PDF or PS report.

```
[TYPE=REPORT, ] PAGESIZE={size|LETTER}, $
```

where:

TYPE=REPORT

Applies the page size to the entire report. Not required, as it is the default.

size

Is the page size. If printing a report, the value should match the size of the paper. Otherwise, the report may be cropped or printed with extra blank space.

Valid values are:

Value	Description
LETTER	8.5 x 11 inches. This value is the default.
LEGAL	8.5 x 14 inches.
TABLOID	11 x 17 inches.
LEDGER	17 x 11 inches.
C	17 x 22 inches.
D	22 x 34 inches.
E	34 x 44 inches.
STATEMENT	5.5 x 8.5 inches.
EXECUTIVE	7.5 x 10.5 inches.
FOLIO	8.5 x 13 inches.
10x14	10 x 14 inches.
A3	297 x 420 millimeters.
A4	210 x 297 millimeters.
A5	148 x 210 millimeters.
B4	250 x 354 millimeters.

Value	Description
B5	182 x 257 millimeters.
QUARTO	215 x 275 millimeters.
ENVELOPE-9	3.875 x 8.875 inches.
ENVELOPE-10	4.125 x 9.5 inches.
ENVELOPE-11	4.5 x 10.375 inches.
ENVELOPE-12	4.5 x 11 inches.
ENVELOPE-14	5 x 11.5 inches.
ENVELOPE-MONARCH	3.875 x 7.5 inches.
ENVELOPE-PERSONAL	3.625 x 6.5 inches.
ENVELOPE-DL	110 x 220 millimeters.
ENVELOPE-C3	324 x 458 millimeters.
ENVELOPE-C4	229 x 324 millimeters.
ENVELOPE-C5	162 x 229 millimeters.
ENVELOPE-C6	114 x 162 millimeters.
ENVELOPE-C65	114 x 229 millimeters.
ENVELOPE-B4	250 x 353 millimeters.
ENVELOPE-B5	176 x 250 millimeters.
ENVELOPE-B6	176 x 125 millimeters.
ENVELOPE-ITALY	110 x 230 millimeters.
US-STANDARD-FANFOLD	14.875 x 11 inches.
GERMAN-STANDARD-FANFOLD	8.5 x 12 inches.
GERMAN-LEGAL-FANFOLD	8.5 x 13 inches.

Syntax: How to Set Page Orientation

This syntax applies to a PDF, PS, or EXL2K report.

```
[TYPE=REPORT, ] ORIENTATION={PORTRAIT|LANDSCAPE}, $
```

where:

TYPE=REPORT

Applies the page orientation to the entire report. Not required, as it is the default value.

PORTRAIT

Displays the report across the narrower dimension of a vertical page, producing a page that is longer than it is wide. PORTRAIT is the default value.

LANDSCAPE

Displays the report across the wider dimension of a horizontal page, producing a page that is wider than it is long.

Syntax: How to Control the Paper Source Used by a PostScript Printer

The SET PSPAGESETUP command includes PostScript code that automatically tells a PostScript printer to set its paper source to the size of paper specified by the PAGESIZE parameter

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

or

```
ON TABLE SET PSPAGESETUP {ON|OFF}
```

where:

ON

Includes PostScript code that tells a PostScript printer to set its paper source to the size of paper specified by the PAGESIZE parameter.

Caution: If you send a job to a printer that does not have the requested paper size loaded, the printer may stop and instruct its operator to load the specified paper. To ensure control over your printing, it is best to set paper size in individual requests so that you can load paper as required.

OFF

Does not include code to coordinate the printer's paper source with the PAGESIZE parameter. OFF is the default value.

Syntax: **How to Set Page Color**

This syntax applies to an HTML report with an internal Cascading Style Sheet.

```
[TYPE=REPORT,] ... PAGECOLOR=color, ... , $
```

where:

```
TYPE=REPORT
```

Applies the color to the entire report. Not required, as it is the default value.

```
color
```

Is a supported color. For a list of supported colors, see [Color Values in a Report](#) on page 522.

Example: **Setting Page Color**

This request sets the page color of an HTML report with internal Cascading Style Sheet to silver.

```
SET HTMLCSS = ON
TABLE FILE CENTORD
ON TABLE SUBHEAD
"SELECTED PRODUCT INVENTORY"
SUM QTY_IN_STOCK/D12 BY PROD_NUM BY SNAME BY STATE
WHERE PROD_NUM EQ '1004'
WHERE SNAME EQ 'eMart'
WHERE STATE EQ 'CA'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, PAGECOLOR=SILVER, GRID=OFF, $
ENDSTYLE
END
```

The output is:



SELECTED PRODUCT INVENTORY			
<u>Product</u>	<u>Store</u>		<u>Quantity</u>
<u>Number#:</u>	<u>Name:</u>	<u>State:</u>	<u>In Stock:</u>
1004	eMart	CA	689,088

Setting Page Margins

How to:

Set the Unit of Measurement

Set Margin Size

Reference:

Page Margin Attributes

You can set the page margins for your report. This includes the top, bottom, left, and right margins. You can also change the default unit of measurement (inches) to either centimeters or points. The unit of measurement applies to page margins, column width, and column position.

Reference: Page Margin Attributes

Attribute	Description	Applies to
UNITS	Sets unit of measurement. Used when specifying margin size or other page characteristics. If you change the current unit of measurement, the new value is applied to all instances in which unit of measurement is used.	PDF PS HTML report with internal Cascading Style Sheet
TOPMARGIN BOTTOMMARGIN LEFTMARGIN RIGHTMARGIN	Sets size of top, bottom, left, and right margin.	PDF PS HTML report with internal Cascading Style Sheet

Syntax: How to Set the Unit of Measurement

This syntax applies to a PDF, PS, or HTML report with internal Cascading Style Sheet.

To set a unit of measurement:

- ❑ In a StyleSheet, add the following attribute:

```
UNITS = units
```

- ❑ Outside of a report request, use:

```
SET UNITS = units
```

- ❑ Within a report request, use

```
ON TABLE SET UNITS units
```

where:

units

Is the unit of measure. Values can be:

[INCHES](#) which specifies the unit of measure as inches. This is the default value.

[CM](#) which specifies the unit of measure as centimeters.

[PTS](#) which specifies the unit of measure as points. Points is a common measurement scale for typefaces.

Syntax: How to Set Margin Size

This syntax applies to a PDF, PS, or HTML report with an internal Cascading Style Sheet.

```
[TYPE=REPORT,] [TOPMARGIN={value|.25},] [BOTTOMMARGIN={value|.25},]  
[LEFTMARGIN={value|.25},] [RIGHTMARGIN={value|.25},] $
```

where:

[TYPE=REPORT](#)

Applies the margin size to the entire report. Not required, as it is the default.

[TOPMARGIN](#)

Sets the top boundary of the report content.

[BOTTOMMARGIN](#)

Sets the bottom boundary of the report content.

[LEFTMARGIN](#)

Sets the left boundary of the report content.

[RIGHTMARGIN](#)

Sets the right boundary of the report content.

value

Is the size of the specified margin. The report content appears inside the margin. If printing a report, specify a value compatible with the printer's print area. For example, if the print area has 0.25 inch margins all around, set the margins to 0.25 inches or larger.

The default value for all margins is 0.25 inches.

Displaying Current Settings: The ? SET STYLE Query

How to:

Display Current Settings

Use the ? SET STYLE query to display the current settings for the STYLESHEET parameter and all page parameters.

Syntax: How to Display Current Settings

The syntax is:

```
? STYLE
```

For example:

```
? style
ONLINE-FMT      STANDARD
OFFLINE-FMT     STANDARD

STYLESHEET      ON
SQUEEZE         OFF
PAGESIZE        Letter
ORIENTATION     PORTRAIT
UNITS           INCHES
LABELPROMPT     OFF
LEFTMARGIN      .250
RIGHTMARGIN     .250
TOPMARGIN       .250
BOTTOMMARGIN    .250
STYLEMODE       FULL
TARGETFRAME
FOCEXURL
BASEURL
```

Note: OFFLINE-FMT is not currently supported. ONLINE-FMT and FOCEXURL apply to WebFOCUS.

Specifying Font Format in a Report

In this section:

Specifying Fonts for Reports

How to:

Specify Font Size in a Report

Specify Bold or Italic Font Style in a Report

Specify Font Color in a Report

Specify Background Color in a Report

Reference:

Usage Notes for Changing Font Size

Color Values in a Report

Using StyleSheet attributes, you can enhance the appearance of a report by specifying the font, its size, and its color. You can also specify the background color of the report. Specifications for background color or font format can be made for a report as a whole, or for headings, footings, and columns designated individually.

Syntax: **How to Specify Font Size in a Report**

To specify a font size, use the following syntax within a StyleSheet.

```
TYPE = type, [subtype,] SIZE=pts, $
```

where:

type

Is the report component you wish to affect, such as REPORT, HEADING, or TITLE.

subtype

Is any additional attribute, such as COLUMN, ACROSS, or ITEM, that is needed to identify the report component that you are formatting. See [Identifying Report Components](#) on page 525 for more information about how to specify different report components.

pts

Is the size of the font in points. The default value is 10, which corresponds to the HTML default font size 3. For more information on the correlation between point size and HTML font size, see [Usage Notes for Changing Font Size](#) on page 519.

Reference: Usage Notes for Changing Font Size

Point size is fixed, except in an HTML report. Relative point size uses a different scale than HTML font size. The following table lists the point size and corresponding HTML font size:

Size in Points	Corresponding HTML Font Size
8 or smaller	1
9	2
<u>10</u>	<u>3</u>
11	4
12	5
13	6
14 or larger	7

Syntax: How to Specify Bold or Italic Font Style in a Report

To specify a font style, use the following syntax within a StyleSheet.

```
TYPE=type, [subtype,] STYLE=[+|-]txtsty[{+|-}txtsty], $
```

where:

type

Is the report component you wish to affect, such as REPORT, HEADING, or TITLE.

subtype

Is any additional attribute, such as COLUMN, ACROSS, or ITEM, that is needed to identify the report component that you are formatting. See [Identifying Report Components](#) on page 525 for more information about how to specify different report components.

txtsty

Is one of the following values: NORMAL, BOLD, ITALIC. The default value is NORMAL.

+

Enables you to specify a combination of font styles. You can add additional font styles to an attribute that already has one or more font styles applied to it.

-

Enables you to remove a font style from an attribute.

Example: Adding and Removing Inherited Font Style in a Report

In the following report request, the font styles bold and italic are specified for the entire report. The inherited italics are removed from the heading, and both styles are removed from the column titles.

The report request is:

```
TABLE FILE GGSALES
HEADING
"Sales Report by Category"
SUM UNITS DOLLARS BY CATEGORY
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, STYLE=BOLD+ITALIC, $
TYPE=HEADING, STYLE=-ITALIC, $
TYPE=TITLE, STYLE=-BOLD-ITALIC, $
TYPE=REPORT, GRID=OFF, $
ENDSTYLE
END
```

The output is:

Sales Report by Category		
<u>Category</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
<i>Coffee</i>	1376266	17231465
<i>Food</i>	1384845	17229344
<i>Gifts</i>	927880	11695502

Syntax: How to Specify Font Color in a Report

To specify a color for the font of a report or a report component, use the following syntax within a StyleSheet.

```
TYPE=type, [subtype,] COLOR={color|RGB(r g b)},$
```

where:

type

Is the report component you wish to affect, such as REPORT, HEADING, or TITLE.

subtype

Is any additional attribute, such as COLUMN, ACROSS, or ITEM, that is needed to identify the report component that you are formatting. See [Identifying Report Components](#) on page 525 for more information about how to specify different report components.

color

Is one of the preset color values such as GREY or GOLD. If the display or output device does not support colors, it substitutes shades of gray. The default value is BLACK. For a complete list of available color values, see [Color Values in a Report](#) on page 522.

RGB

Specifies the font color using a mixture of red, green, and blue.

(r g b)

Is the desired intensity of red, green, and blue, respectively. The values are on a scale of 0 to 255, where 0 is the least intense and 255 is the most intense. Note that using the three color components in equal intensities results in shades of gray.

Syntax: How to Specify Background Color in a Report

Use the following StyleSheet syntax to specify a color for the background of a report.

Note that when using BACKCOLOR in a PDF report, extra space is added to the top, bottom, right, and left of each cell of data in the report. This is for readability and to prevent truncation.

```
TYPE=type, [subtype,] BACKCOLOR={color|RGB(r g b)}, $
```

where:

type

Is the report component you wish to affect, such as REPORT, HEADING, or TITLE.

subtype

Is any additional attribute, such as COLUMN, ACROSS, or ITEM, that is needed to identify the report component that you are formatting. See [Color Values in a Report](#) on page 522 for more information about how to specify different report components.

color

Is the background color, which fills the space of the specified report component. The default value is NONE. If you are creating a report in HTML format, background colors will only appear in Web browsers that support them. Netscape Navigator 3.0 and higher and Microsoft Internet Explorer 3.0 and higher support background colors.

RGB

Specifies the font color using a mixture of red, green, and blue.

(r g b)

Is the desired intensity of red, green, and blue, respectively. The values are on a scale of 0 to 255, where 0 is the least intense and 255 is the most intense. Note that using the three color components in equal intensities results in shades of gray.

Reference: Color Values in a Report

The following chart lists all available color values that can be utilized with the syntax.

`COLOR=color`, `BACKCOLOR=color`, or `PAGECOLOR=color`

where *color* is one of the following values:

AQUA (CYAN)	MEDIUM FOREST GREEN (OLIVE)
AQUAMARINE	MEDIUM GOLDENROD
BLACK	MEDIUM ORCHID
BLUE VIOLET	MEDIUM SLATE BLUE
CADET BLUE	MEDIUM SPRING GREEN
CORAL	MEDIUM TURQUOISE
CORNFLOWER BLUE	MEDIUM VIOLET RED
CYAN (AQUA)	MIDNIGHT BLUE
DARK GREEN	NAVY (NAVY BLUE)
DARK OLIVE GREEN	OLIVE (MEDIUM FOREST GREEN)
DARK ORCHID	ORANGE
DARK SLATE BLUE (PURPLE)	ORANGE RED
DARK SLATE GREY	ORCHID
DARK TURQUOISE	PALE GREEN
DIM GREY (GRAY, GREY)	PINK
FIREBRICK	PLUM
FOREST GREEN (GREEN)	PURPLE (DARK SLATE BLUE)
FUCHSIA (MAGENTA)	RED
GOLD	SALMON
GOLDENROD	SEA GREEN
GRAY (DIM GREY, GREY)	SIENNA
GREEN (FOREST GREEN)	SILVER
GREEN YELLOW	SKY BLUE
GREY (DIM GREY, GRAY)	SLATE BLUE
INDIAN RED	STEEL BLUE (TEAL)
KHAKI	TAN
LIGHT BLUE	TEAL (STEEL BLUE)
LIGHT GREY	THISTLE
LIGHT STEEL BLUE	TURQUOISE
LIME	VIOLET
LIME GREEN	VIOLET RED
MAGENTA (FUCHSIA)	WHEAT
MAROON	WHITE
MEDIUM AQUAMARINE	YELLOW
MEDIUM BLUE	YELLOW GREEN

Note that some colors may not appear as specified in the Excel formats, because FOCUS has no control over how Excel renders colors.

Specifying Fonts for Reports

How to:

Specify Fonts in a Report

Specify the Default Browser Fonts for HTML Reports

You can specify your own fonts in a report by using the FONT attribute in a StyleSheet. If you are specifying a font for an HTML report, the user's Web browser must support the font. If the Web browser does not support the font, it reverts to its default behavior of using the default proportional font.

Syntax: **How to Specify Fonts in a Report**

To specify a font for your report, use the following syntax within a StyleSheet.

```
TYPE=type, [subtype,] FONT='font[font]', $
```

where:

type

Is the report component you wish to affect, such as REPORT, HEADING, or TITLE.

subtype

Is any additional attribute, such as COLUMN, ACROSS, or ITEM, that is needed to identify the report component that you are formatting. See [Identifying Report Components](#) on page 525 for more information about how to specify different report components.

font

Is the name of the font. You must enclose the value in single quotes. If you are creating an HTML report, you can specify more than one font within the single quotes to accommodate more than one browser.

Note: In an HTML report, specifying different fonts for several different report components significantly increases the size of the source code.

Example: **Specifying Multiple Fonts in an HTML Report**

To control how a report looks on more than one platform, you can specify both a common Windows font and a common UNIX font in a request. The Web browser searches for the first font in the list. If the browser does not find this font, it searches for the next one in the list. If none of the fonts are identified, the browser uses the default proportional font.

In this example, the Web browser first searches for the Arial font. If the browser does not find Arial, it searches for the Helvetica font. If neither font is identified, the browser uses the default proportional font.

```
TYPE=REPORT, FONT='ARIAL,HELVETICA', $
```

Syntax: How to Specify the Default Browser Fonts for HTML Reports

A browser assigns specific fonts as the default proportional and default monospace fonts. To specify a default browser font for an HTML report, you use the reserved names, DEFAULT-PROPORTIONAL and DEFAULT-FIXED in the StyleSheet of your report. The browser displays the report accordingly.

To select the default fixed or proportional font of the browser, use the following syntax. Note that you must specify TYPE to indicate which report components you wish to affect.

```
FONT={DEFAULT-PROPORTIONAL|DEFAULT-FIXED}, $
```

where:

`DEFAULT-PROPORTIONAL`

Specifies the default proportional font of the Web browser.

`DEFAULT-FIXED`

Specifies the default monospace font of the Web browser.

Example: Specifying Default Browser Fonts

In this example, the Web browser uses the default monospace font for the entire report except the report heading and column headings. For these headings, the Web browser uses the default proportional font.

```
TABLE FILE GGSALES
HEADING
"Sales Report"
ON TABLE HOLD FORMAT HTML
SUM UNITS DOLLARS BY CATEGORY BY PRODUCT
ON TABLE SET STYLE *
TYPE=REPORT, FONT=DEFAULT-FIXED, $
TYPE=TITLE, FONT=DEFAULT-PROPORTIONAL, $
TYPE=HEADING, FONT=DEFAULT-PROPORTIONAL, $
TYPE=REPORT, GRID=OFF, $
ENDSTYLE
END
```

The output is:

Sales Report

<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
Coffee	Capuccino	189217	2381600
	Espresso	308986	3906243
	Latte	878063	10943622
Food	Biscotti	421377	5263328
	Croissant	630054	7749902
	Scone	333414	4216114
Gifts	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829



Identifying Report Components

In this section:

Identifying an Entire Report, Column, or Row

Identifying Data

Identifying Totals and Subtotals

Identifying a Heading, Footing, Title, or FML Free Text

Identifying a Column or Row Title

Identifying a Heading or Footing

Identifying a Page Number, Underline, or Skipped Line

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 [FOCUS StyleSheet Attribute Inheritance](#) on page 581.

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. See Identifying an Entire Report, Column, or Row on page 527.
PAGENUM	Default page numbers. See Identifying a Page Number, Underline, or Skipped Line on page 565. 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. See Identifying a Heading or Footing on page 554.
TABFOOTING	A footing on or after the last page of a report, generated by ON TABLE SUBFOOT. See Identifying a Heading or Footing on page 554.
HEADING	Headings at the top of each report page. See Identifying a Heading or Footing on page 554.
FOOTING	Footings at the bottom of each report page. See Identifying a Heading or Footing on page 554.
SUBHEAD	Headings before a particular sort field, generated by ON sortfield SUBHEAD. See Identifying a Heading or Footing on page 554.
SUBFOOT	Footings after a particular sort field, generated by ON sortfield SUBFOOT. See Identifying a Heading or Footing on page 554.
DATA	Report data. See Identifying Data on page 537.
TITLE	Column titles. See Identifying a Column or Row Title on page 549.
ACROSSTITLE	ACROSS field names (that is, field names used in ACROSS phrases). See Identifying a Heading, Footing, Title, or FML Free Text on page 548
ACROSSVALUE	ACROSS field values (that is, values of the ACROSS field). These values become column titles in the report. See Identifying Data on page 537.

TYPE	Report Component
SUBTOTAL	Totals generated by SUBTOTAL, SUB-TOTAL, RECOMPUTE, and SUMMARIZE. See Identifying Totals and Subtotals on page 542.
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. See Identifying Totals and Subtotals on page 542.
RECAP	Lines generated by ON field name RECAP or ON field name COMPUTE. See Identifying Totals and Subtotals on page 542.
UNDERLINE	Underlines generated by ON field name UNDER-LINE. See Identifying a Page Number, Underline, or Skipped Line on page 565.
SKIPLINE	Skipped lines generated by ON field name SKIP-LINE. See Identifying a Page Number, Underline, or Skipped Line on page 565.
FREETEXT	FML free text. See Identifying a Heading, Footing, Title, or FML Free Text on page 548.

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). For example, to choose the third column for the entire report, use the parameters:

- ❑ TYPE=REPORT
- ❑ COLUMN=3

Identifying an Entire Report, Column, or Row

How to:

- Identify the Entire Report
- Identify an Entire Column
- Identify an Entire Financial Modeling Language (FML) Row
- Identify an Entire Total or Subtotal Row

You can apply formatting to an:

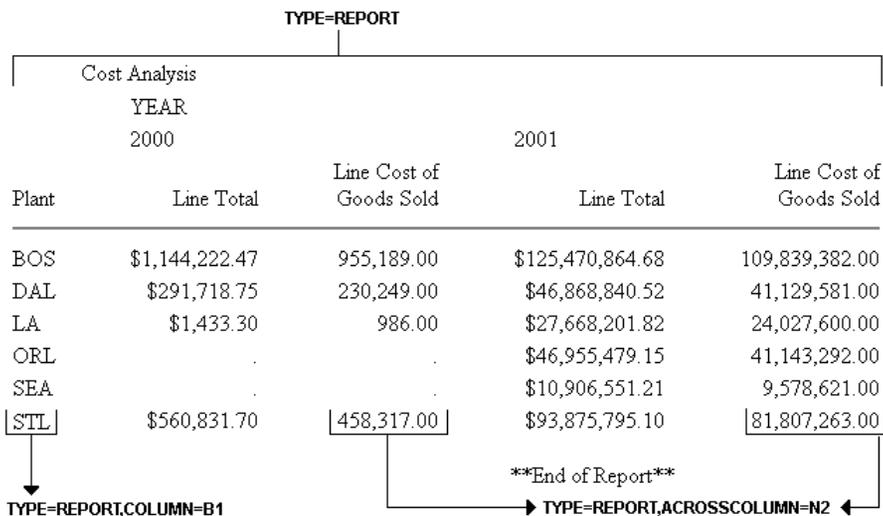
- ❑ **Entire report.** For more information, see [How to Identify the Entire Report](#) on page 529.

- ❑ **Entire column** within a report, both its title and data (including ROW-TOTAL columns). For more information, see [How to Identify an Entire Column](#) on page 529.
- ❑ **Entire row** within a report"either an FML (Financial Modeling Language) row, or a total or subtotal row"comprising the row's labeling text and its data. For more information, see [How to Identify an Entire Financial Modeling Language \(FML\) Row](#) on page 533, and [How to Identify an Entire Total or Subtotal Row](#) on page 534.

You can also identify an entire horizontal sort (ACROSS) title or value row in a StyleSheet, although each of these rows contains only a single kind of information. For details, see [How to Identify a Column Title](#) on page 550.

The following illustrates where the REPORT component and the COLUMN and ACROSSCOLUMN attributes appear in a report, and which TYPE values you use to identify them. Although in this example the value for COLUMN is B1 and the value for ACROSSCOLUMN is N2, these are not the only values you can use to identify these components.

```
TABLE FILE CENTORD
SUM LINEPRICE LINE_COGS AS 'Line Cost of,Goods Sold'
BY PLANT AS 'Plant'
ACROSS YEAR
WHERE YEAR EQ 2000 OR 2001
HEADING
"Cost Analysis"
FOOTING CENTER
"***End of Report***"
ON TABLE SET PAGE-NUM OFF
END
```



Note: Since this request simply illustrates where the components appear in a report, it omits a StyleSheet.

Syntax: **How to Identify the Entire Report**

To select the entire report, use the syntax:

```
TYPE = REPORT
```

Syntax: **How to Identify an Entire Column**

```
TYPE=REPORT, coltype=column
```

where:

coltype

Specifies the type of column. It can be:

COLUMN specifies a sort column (generated by BY), a display column (generated by PRINT, LIST, SUM, or COUNT), a computed column (generated by COMPUTE), or a column of row totals (generated by ROW-TOTAL).

ACROSSCOLUMN specifies every instance of a column that is repeated across a horizontal sort (ACROSS) row.

column

Specifies one or more columns. If you are identifying an ACROSSCOLUMN, the only valid identifiers are N_n and P_n .

Options for identifying columns in a StyleSheet are:

Identifier	Description
N_n	Identifies a column by its position in the report. To determine this value, count vertical sort (BY) fields, display fields, and ROW-TOTAL fields, from left to right, including NOPRINT fields. For an example, see How to Identify a Column of Data on page 540.
P_n	Identifies a column by its position in the report. To determine the value of n , count vertical sort (BY) fields, display fields, and ROW-TOTAL fields from left to right. Do not count NOPRINT fields.

Identifier	Description
<i>Cn</i>	<p>Identifies a display column by its position in the report. To determine the value of <i>n</i>, count only display fields from left to right, including NOPRINT fields. Do not count vertical sort (BY) fields or ROW-TOTAL fields.</p> <p>To select all display fields use C*.</p>
<i>Bn</i>	<p>Identifies a vertical sort (BY) column by its position in the report. To determine the value of <i>n</i>, count only vertical sort (BY) fields, including NOPRINTs, from left to right.</p> <p>To select all BY fields use B*.</p>
<i>field</i>	<p>Identifies a column by its field name.</p> <p>When a field occurs more than once, use <i>field(n)</i> to select a particular occurrence or <i>field(*)</i> to select all occurrences of the field.</p>
ROWTOTAL	<p>Identifies a column of row totals generated using ROW-TOTAL. When used with ACROSS and multiple display commands, ROWTOTAL generates multiple total columns. Use ROWTOTAL(<i>n</i>) to select a particular total column. Use ROWTOTAL(<i>field</i>) to select the row total column for a particular field.</p> <p>Use ROWTOTAL(*) to select all row total columns in the report.</p>

Note: Within a StyleSheet, all columns must be specified in the same way, either by field name or positional reference.

Example: Identifying an Entire Column

The following illustrates how to identify an entire column, which consists of the column data and the column title, in a report. The relevant StyleSheet declaration is highlighted in the request.

Note: To produce the same results you can, alternatively, use the values P1, B1, or the field name (PRODNAME) for the COLUMN attribute in the StyleSheet declaration.

```
TABLE FILE CENTINV
HEADING
"Excess Stock Report"
SUM QTY_IN_STOCK
BY PRODNAME
WHERE QTY_IN_STOCK GT 10000
FOOTING CENTER
"***End of Report***"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
TYPE=REPORT, COLUMN=N1, STYLE=ITALIC,$
ENDSTYLE
END
```

The output is:

Excess Stock Report	
<i>Product</i>	Quantity
<i><u>Name:</u></i>	<i><u>In Stock:</u></i>
<i>AR2 35MM Camera 8 X</i>	11499
<i>AR3 35MM Camera 10 X</i>	12444
<i>Combo Player - 4 Hd VCR + DVD</i>	13527
<i>QX Portable CD Player</i>	22000
<i>ZC Digital PDA - Standard</i>	33000
<i>ZT Digital PDA - Commercial</i>	21000
<i>2 Hd VCR LCD Menu</i>	43068
<i>250 8MM Camcorder 40 X</i>	60073
<i>330DX Digital Camera 1024K P</i>	12707
<i>750SL Digital Camcorder 300 X</i>	10758

End of Report

Example: Identifying an Entire Horizontal (ACROSS) Column

The following illustrates how to identify a horizontal (ACROSS) column. When you identify and format an ACROSSCOLUMN, all data values and the column title sort. The relevant StyleSheet declarations are highlighted in the request.

Note: To produce the same results you can alternatively use the values P1 and P2, respectively, for the ACROSSCOLUMN attribute.

```
TABLE FILE CENTORD
SUM LINEPRICE LINE_COGS AS 'Line Cost of,Goods Sold'
BY PLANT AS 'Plant'
ACROSS YEAR
WHERE YEAR EQ 2000 OR 2001
HEADING
"Cost Analysis"
FOOTING CENTER
"***End of Report***"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
TYPE=REPORT, ACROSSCOLUMN=N1, STYLE=ITALIC,$
TYPE=REPORT, ACROSSCOLUMN=N2, STYLE=BOLD,$
ENDSTYLE
END
```

The output is:

Cost Analysis				
YEAR				
2000		2001		
Plant	<i>Line Total</i>	Line Cost of Goods Sold	<i>Line Total</i>	Line Cost of Goods Sold
BOS	\$1,144,222.47	955,189.00	\$125,470,864.68	109,839,382.00
DAL	\$291,718.75	230,249.00	\$46,868,840.52	41,129,581.00
LA	\$1,433.30	986.00	\$27,668,201.82	24,027,600.00
ORL	.	.	\$46,955,479.15	41,143,292.00
SEA	.	.	\$10,906,551.21	9,578,621.00
STL	\$560,831.70	458,317.00	\$93,875,795.10	81,807,263.00

****End of Report****

Syntax: How to Identify an Entire Financial Modeling Language (FML) Row

`TYPE=REPORT, LABEL=label`

where:

Rn

Is an implicit row label. To determine the value of *n*, count the number of rows up to and including the desired row.

label

Is an explicit row label.

Example: Identifying an Entire FML Row

The following illustrates how to identify an entire FML row, consisting of the row label and the row data. The relevant StyleSheet declarations are highlighted in the request.

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND' LABEL COH OVER
1020 AS 'DEMAND DEPOSITS' LABEL DD OVER
1030 AS 'TIME DEPOSITS' LABEL TD OVER
BAR OVER
RECAP TOTCASH = R1 + R2 + R3; AS 'TOTAL CASH'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=REPORT, LABEL=COH, STYLE=ITALIC, $
TYPE=REPORT, LABEL=DD, STYLE=ITALIC, $
TYPE=REPORT, LABEL=TD, STYLE=ITALIC, $
ENDSTYLE
END
```

The output is:

	<u>AMOUNT</u>
<i>CASH ON HAND</i>	<i>8,784</i>
<i>DEMAND DEPOSITS</i>	<i>4,494</i>
<i>TIME DEPOSITS</i>	<i>7,961</i>

TOTAL CASH	21,239

Syntax: How to Identify an Entire Total or Subtotal Row

```
TYPE=type, [BY=sortcolumn]
```

where:

type

Identifies a subtotal or total. Select from:

GRANDTOTAL which is a grand total (generated by COLUMN-TOTAL, SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE).

SUBTOTAL which is a subtotal (generated by SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE).

`RECAP` which is a subtotal calculation (generated by `ON sortfield RECAP` or `ON sortfield COMPUTE`).

`BY`

When there are several subtotal commands, each associated with a different vertical sort (`BY`) column, this enables you to identify which of the subtotal commands you wish to format.

`sortcolumn`

Specifies the vertical sort (`BY`) column associated with one of a report's several subtotal commands. Use the field name to identify the sort column.

Example: Identifying an Entire Total Row

The following illustrates how to identify an entire `COLUMN-TOTAL` row in a `StyleSheet`. The relevant `StyleSheet` declaration is highlighted in the request.

```
TABLE FILE SALES
SUM RETURNS DAMAGED AND ROW-TOTAL AND COLUMN-TOTAL
BY PROD_CODE
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=GRANDTOTAL, STYLE=BOLD, SIZE=12, $
ENDSTYLE
END
```

The output is:

<u>PROD_CODE</u>	<u>RETURNS</u>	<u>DAMAGED</u>	<u>TOTAL</u>
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: Identifying a Row Total

The following illustrates how to identify a row total. The relevant StyleSheet declaration is highlighted in the request. Note that if you want to format an instance of row-total, you can add a WHEN statement to your StyleSheet. For details, see [Conditionally Formatting in a StyleSheet](#) on page 585.

```
TABLE FILE SALES
SUM RETURNS DAMAGED AND ROW-TOTAL
BY PROD_CODE AS 'PRODUCT, CODE'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PS
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=REPORT, COLUMN=ROWTOTAL, STYLE=BOLD, $
ENDSTYLE
END
```

The output is:

<u>PRODUCT</u> <u>CODE</u>	<u>RETURNS</u>	<u>DAMAGED</u>	<u>TOTAL</u>
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

Identifying Data

How to:

Identify All Data

Identify a Column of Data

Identify a Row of Horizontal Sort (ACROSS) Data

You can identify and format many categories of data in a report, including:

- ❑ **All of a report's data.** For more information, see [How to Identify All Data](#) on page 538.
- ❑ **Columns of data**, including sort columns and display columns. For more information, see [How to Identify a Column of Data](#) on page 540.
- ❑ **Sort rows** (that is, ACROSS field values). For more information, see [How to Identify a Row of Horizontal Sort \(ACROSS\) Data](#) on page 541.
- ❑ **Totals and subtotals.** For more information, see [Identifying Totals and Subtotals](#) on page 542.

The following illustrates where the DATA and ACROSSVALUE components appear in a report, and which TYPE values you use to identify them.

```
TABLE FILE CENTORD
HEADING CENTER
"UNITS SOLD IN 2002 BY PLANT"
SUM QUANTITY AND ROW-TOTAL AS '2002 TOTAL'
ACROSS QUARTER
BY PLANTLNG AS 'PLANT'
WHERE YEAR EQ 2002
ON TABLE COLUMN-TOTAL AS 'TOTAL UNITS'
ON TABLE SET PAGE-NUM OFF
END
```

		UNITS SOLD IN 2002 BY PLANT				
		QUARTER				2002 TOTAL
TYPE=ACROSSVALUE	PLANT	Q1	Q2	Q3	Q4	
TYPE=DATA	Boston	181,055	138,752	27,103	83,240	430,150
	Dallas	31,495	53,141	34,675	35,743	155,054
	Los Angeles	42,752	25,232	44,240	29,926	142,150
	Orlando	33,375	47,691	27,635	42,873	151,574
	Seattle	14,509	24,538	4,532	4,560	48,139
	St Louis	103,571	30,669	50,335	29,584	214,159
	TOTAL UNITS	406,757	320,023	188,520	225,926	1,526,451

Note: Since this request simply illustrates where the components appear in a report, it omits a StyleSheet.

Syntax: **How to Identify All Data**

To identify all report data in a StyleSheet"except for column totals, grand totals, subtotals, and horizontal sort (ACROSS) values, which need to be identified separately" use this attribute and value:

```
TYPE = DATA
```

Example: Identifying All Data in a Report

The following illustrates how to identify all of the data in a report. The relevant StyleSheet declaration is highlighted in the request.

```
TABLE FILE CENTORD
HEADING CENTER
"UNITS SOLD IN 2002 BY PLANT"
SUM QUANTITY AND ROW-TOTAL AS '2002 TOTAL'
ACROSS QUARTER
BY PLANTING AS 'PLANT'
WHERE YEAR EQ 2002
ON TABLE COLUMN-TOTAL AS 'TOTAL UNITS'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=DATA, STYLE=BOLD, $
ENDSTYLE
END
```

In the output, the titles and grand total are not bolded because they are not data values:

UNITS SOLD IN 2002 BY PLANT					
QUARTER					
PLANT	Q1	Q2	Q3	Q4	2002 TOTAL
Boston	165,938	202,100	84,571	86,227	538,836
Dallas	62,693	63,403	35,747	42,094	203,937
Los Angeles	55,542	30,326	17,188	16,869	119,925
Orlando	76,261	57,639	32,595	35,895	202,390
Seattle	16,887	17,496	5,454	5,512	45,349
St Louis	150,293	109,284	65,161	80,530	405,268
TOTAL UNITS	527,614	480,248	240,716	267,127	1,515,705

Syntax: How to Identify a Column of Data

TYPE=DATA, COLUMN=*column*

where:

column

Specifies one or more columns that you wish to format. For a list of values, see [How to Identify an Entire Column](#) on page 529.

Example: Identifying a Column of Data

The following illustrates how to identify a column of data. The relevant StyleSheet declaration is highlighted in the request.

Note that when identifying a column using *Nn*, NOPRINT columns are counted. Even though the Product Name field is the first column in this report, it is identified with N2 because of the NOPRINT column.

```
TABLE FILE CENTORD
PRINT QUANTITY LINEPRICE LINE_COGS
BY ORDER_NUM NOPRINT
BY PRODNAME
WHERE ORDER_NUM EQ '48045'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
TYPE=DATA, COLUMN=N2, STYLE=ITALIC,$
ENDSTYLE
END
```

The output is:

<u>Product Name:</u>	<u>Quantity:</u>	<u>Line Total</u>	<u>Cost Of Goods Sold</u>
<i>DVD Upgrade Unit for Cent. VCR</i>	121	\$18,177.77	16,819.00
<i>QX Portable CD Player</i>	266	\$33,847.80	26,334.00
<i>110 VHS-C Camcorder 20 X</i>	266	\$70,901.90	66,234.00
<i>120 VHS-C Camcorder 40 X</i>	266	\$76,511.09	68,894.00
<i>150 8MM Camcorder 20 X</i>	121	\$28,473.78	29,040.00

Syntax: How to Identify a Row of Horizontal Sort (ACROSS) Data

```
TYPE=ACROSSVALUE, [ACROSS={fieldname|Nn}]
```

where:

ACROSS

If you have a request with multiple ACROSS fields, you can identify each field using the ACROSS identifier. You only need to include the ACROSS identifier if you have multiple ACROSS fields in your request.

fieldname

Specifies a horizontal sort row by its field name.

Nn

Specifies a horizontal sort row by its position in the sequence of horizontal sort rows.

Example: Identifying a Row of Horizontal Sort (ACROSS) Data

The following illustrates how to identify a row of horizontal data values. The relevant StyleSheet declaration is highlighted in the request.

```
TABLE FILE CENTORD
HEADING
"Units Sold"
SUM QUANTITY
BY PRODNAME
ACROSS PLANT AS 'Manufacturing Plant'
WHERE PRODTYPE EQ 'Digital'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=HEADING, SIZE=12, $
TYPE=ACROSSVALUE, ACROSS=PLANT, STYLE=BOLD, $
ENDSTYLE
END
```

The output is:

Units Sold

Product Name:	Manufacturing Plant					
	BOS	DAL	LA	ORL	SEA	STL
Combo Player - 4 Hd VCR + DVD	27,687	15,985	9,634	13,816	2,359	22,097
DVD Upgrade Unit for Cent. VCR	47,112	25,602	16,152	23,429	4,539	32,875
QX Portable CD Player	108,696	35,355	26,215	46,388	8,056	83,564
R5 Micro Digital Tape Recorder	188,384	73,255	41,169	63,120	15,385	142,714
ZC Digital PDA - Standard	45,890	13,154	4,664	10,249	6,439	30,213
ZT Digital PDA - Commercial	181,750	70,111	41,149	61,244	15,039	130,942
330DX Digital Camera 1024K P	3,949	2,767	562	561	2	1,132
650DL Digital Camcorder 150 X	44,930	14,548	7,995	17,972	4,967	31,436
750SL Digital Camcorder 300 X	4,098	2	6	283	642	1,644

Note: To produce the same results you can alternatively use the value N1 for the ACROSS attribute in the StyleSheet declaration. For example, TYPE=ACROSSVALUE, ACROSS=N1, STYLE=BOLD, \$.

Identifying Totals and Subtotals

How to:

Identify a Grand Total, Subtotal, or Subtotal Calculation

Within a StyleSheet, you can identify a report's grand totals, subtotals, subtotal calculations (generated by ON *sortfield* RECAP or ON *sortfield* COMPUTE), column totals, and row totals in order to format them. For details on identifying row totals, see [Identifying an Entire Report, Column, or Row](#) on page 527.

The following example illustrates where these components are in a report, and which TYPE values you use to identify them.

```
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
ON DEPARTMENT SUBTOTAL
END
```

DEPARTMENT	PAY_DATE	DED_AMT	GROSS	
MIS	81/11/30	\$1,406.79	\$2,147.75	
	81/12/31	\$1,406.79	\$2,147.75	
*TOTAL MIS		\$2813.58	\$4295.50	← TYPE = SUBTOTAL
** DEPT_NET		\$2,311.98		← TYPE = RECAP
PRODUCTION	81/11/30	\$141.66	\$833.33	
	82/01/29	\$1,560.09	\$3,705.84	
*TOTAL PRODUCTION		\$1701.75	\$4539.17	← TYPE = SUBTOTAL
** DEPT_NET		\$2,531.14		← TYPE = RECAP
TOTAL		\$5,578.35	\$10,421.47	← TYPE = GRANDTOTAL

Note: Since this request simply illustrates how to identify different types of totals and subtotals, it omits a StyleSheet.

Syntax: How to Identify a Grand Total, Subtotal, or Subtotal Calculation

```
TYPE=type, [BY=sortfield] [coltype=column]
```

where:

type

Identifies a subtotal or total. Select from:

GRANDTOTAL which is a grand total (generated by COLUMN-TOTAL, SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE).

SUBTOTAL which is a subtotal (generated by SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE).

RECAP which is a subtotal calculation (generated by `ON sortfield RECAP` or `ON sortfield COMPUTE`).

BY

If you have requests with multiple BY fields, and two or more have subtotal commands associated with them, you can identify each field using the BY identifier. This is helpful when you want to format each subtotal differently or when you want to format only one subtotal.

You need to include the BY identifier only if you have multiple BY fields in your request.

sortfield

Specifies the BY field associated with one of a report's several subtotal commands. Use the fieldname for the value (`BY=fieldname`).

coltype

Identifies a specific column for formatting. When you include the **COLUMN** or **ACROSSCOLUMN** identifier in your declaration, only the subtotal *values* receive the formatting; the labeling text will not. Values can be:

COLUMN which is a display column (generated by `PRINT`, `LIST`, `SUM`, or `COUNT`) or a computed column (generated by `COMPUTE`).

ACROSSCOLUMN where every instance of a display or computed column that is repeated across a horizontal sort (**ACROSS**) row.

If there are several columns being totaled or subtotaled by one command, and you do not specify a column in the StyleSheet, the formatting will be applied to the totals or subtotals for *all* of the columns. It will also be applied to the labeling text for the total and subtotal values.

column

Specifies the column whose totals or subtotals you wish to format. For a list of values, see [How to Identify an Entire Column](#) on page 529.

Example: Identifying Subtotals and the Grand Total

The following illustrates how to identify subtotals and the grand total in a report request. In this example, only subtotal values in the **QUANTITY** and **LINE_COGS** fields are formatted, so the **COLUMN** attribute is included in the StyleSheet declarations.

The grand total in this request is generated by **COLUMN-TOTAL**.

Since there are two SUBTOTAL commands associated with two of the three BY fields (PLANT and ORDER_NO), the BY attribute is also included in each declaration to ensure the formatting is applied to the correct value.

```
TABLE FILE CENTORD
SUM QUANTITY LINE_COGS AS 'Line Cost of, Goods Sold'
BY PLANT AS 'Plant'
BY ORDER_NUM AS 'Order,Num'
BY PRODNAME
ON PLANT SUBTOTAL AS 'Total:'
ON ORDER_NUM SUBTOTAL AS 'Total:'
WHERE ORDER_NUM EQ '35774' OR '48041'
WHERE PLANT EQ 'BOS'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
TYPE=SUBTOTAL, BY=PLANT, COLUMN=LINE_COGS, STYLE=BOLD+ITALIC, COLOR=BLUE,$
TYPE=SUBTOTAL, BY=ORDER_NUM, COLUMN=QUANTITY, STYLE=BOLD, SIZE=11,$
TYPE=GRANDTOTAL, COLUMN=QUANTITY, STYLE=ITALIC, SIZE=11,$ENDSTYLE
END
```

The output is:

<u>Plant Num</u>	<u>Order Product Name:</u>	<u>Quantity:</u>	<u>Line Cost of Goods Sold</u>
BOS 35774	110 VHS-C Camcorder 20 X	146	36,354.00
	120 VHS-C Camcorder 40 X	146	37,814.00
	2 Hd VCR LCD Menu	279	35,991.00
	650DL Digital Camcorder 150 X	146	103,660.00
Total: 35774		717	213,819.00
48041	Combo Player - 4 Hd VCR + DVD	1	289.00
	DVD Upgrade Unit for Cent. VCR	1	139.00
	R5 Micro Digital Tape Recorder	1	69.00
	ZT Digital PDA - Commercial	1	349.00
	150 8MM Camcorder 20 X	1	240.00
Total: 48041		5	1,086.00
Total: BOS		722	214,905.00

Note:

- ❑ To style the entire grand total row, remove the COLUMN attribute from the StyleSheet declaration.
- ❑ To produce the same results, you can alternatively use the values N5, P5, or C3 for the COLUMN attribute in the StyleSheet declaration.
- ❑ To style an entire subtotal row, remove the COLUMN and BY attributes from the StyleSheet declaration.
- ❑ To produce the same results you can, alternatively, use the values COLUMN=N6, COLUMN=P6, or COLUMN=C3 for the COLUMN=LINE_COGS attribute.
- ❑ To produce the same results you can, alternatively, use the values COLUMN=N4, COLUMN=P4, or COLUMN=C1 for the COLUMN=QUANTITY attribute.

Example: Identifying a Subtotal Calculation (RECAP/COMPUTE)

The following illustrates how to identify a subtotal calculation created with a RECAP or COMPUTE phrase. In this example, the subtotal calculation is generated with ON PLANT RECAP QTY/F6=QUANTITY. The relevant StyleSheet declaration is highlighted in the request.

Note: If there are multiple RECAP or COMPUTE fields in your request, you can distinguish them by adding BY=*fieldname* to the StyleSheet declaration.

```
TABLE FILE CENTORD
SUM QUANTITY LINEPRICE LINE_COGS AS 'Line Cost of, Goods Sold'
BY PLANT AS 'Plant' BY ORDER_NUM
ON PLANT RECAP QTY/F6=QUANTITY;
WHERE PLANT EQ 'BOS'
WHERE ORDER_NUM GT '60000' AND ORDER_NUM LT '70000'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=RECAP, STYLE=BOLD+ITALIC, $
ENDSTYLE
END
```

The output is:

<u>Plant</u>	<u>Order</u> <u>Number:</u>	<u>Quantity:</u>	<u>Line</u> <u>Total</u>	<u>Line</u> <u>Cost of</u>	<u>Goods</u> <u>Sold</u>
BOS	68290	1,306	\$304,816.76	268,366.00	
	68327	5	\$1,923.45	1,456.00	
	68580	1,015	\$236,729.89	209,487.00	
	68710	4	\$994.44	757.00	
	68900	11	\$3,760.44	2,617.00	

**** QTY 2341**

Example: Styling Multiple RECAP Statements in a Matrix

You can style multiple RECAP commands in a matrix when the RECAP statements are placed after the last ACROSS value:

```
TABLE FILE GGSALES
SUM UNITS
BY PRODUCT
ACROSS REGION
RECAP
TTL1/I8=C1+C2+C3+C4;
TTL2/D12.2=TTL1*1.25;
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=DATA, COLUMN=TTL1 (*), COLOR=BLUE, BACKCOLOR=SILVER, STYLE=BOLD, $
TYPE=DATA, COLUMN=TTL2 (*), COLOR=RED, BACKCOLOR=AQUA, STYLE=BOLD, $
ENDSTYLE
END
```

The output is:

Product	Region				TTL1	TTL2
	Midwest	Northeast	Southeast	West		
Biscotti	86105	145242	119594	70436	421377	526,721.25
Capuccino	.	44785	73264	71168	189217	236,521.25
Coffee Grinder	50393	40977	47083	48081	186534	233,167.50
Coffee Pot	47156	46185	49922	47432	190695	238,368.75
Croissant	139182	137394	156456	197022	630054	787,567.50
Espresso	101154	68127	68030	71675	308986	386,232.50
Latte	231623	222866	209654	213920	878063	1,097,578.75
Mug	86718	91497	88474	93881	360570	450,712.50
Scone	116127	70732	73779	72776	333414	416,767.50
Thermos	46587	48870	48976	45648	190081	237,601.25

Identifying a Heading, Footing, Title, or FML Free Text

A report's data is framed by headings, footings, and titles; these provide context for the data. You can identify and format many categories of headings, footings, and titles in a report, including:

- Report, page, and sort headings.
- Report, page, and sort footings.

- ❑ Column titles.
- ❑ Horizontal sort (ACROSS) titles and values.
- ❑ Free text in Financial Modeling Language (FML) reports.

You can also use a StyleSheet to create a report title that:

- ❑ Overrides the default report title (FOCUS Report) that appears in your browser's title bar in an HTML report.
- ❑ Replaces the default worksheet tab name with the name specified in an EXL2K report.

Identifying a Column or Row Title

How to:

Identify a Column Title

Identify a Horizontal Sort Title or Value

Identify Free Text in an FML Report

Create a Custom Report Title

Within a StyleSheet, you can identify a report's column titles and horizontal sort (ACROSS) values to format. The following example illustrates where column titles and horizontal sort values are stored in a report, and which TYPE values you use to identify them.

```
TABLE FILE EMPLOYEE
SUM GROSS AND DED_AMT
ACROSS DEPARTMENT BY PAY_DATE
END
```

PAGE 1

PAY_DATE	DEPARTMENT ←		PRODUCTION ←	
	MIS		GROSS	DED_AMT ←
	GROSS	DED_AMT	GROSS	DED_AMT
81/11/30	\$2,147.75	\$1,406.79	\$833.33	\$141.66
81/12/31	\$2,147.75	\$1,406.79	\$833.33	\$141.66
82/01/29	\$3,247.75	\$1,740.89	\$3,705.84	\$1,560.09
82/02/26	\$3,247.75	\$1,740.89	\$4,959.84	\$2,061.69
82/03/31	\$3,247.75	\$1,740.89	\$4,959.84	\$2,061.69
82/04/30	\$5,890.84	\$3,386.73	\$4,959.84	\$2,061.69
82/05/28	\$6,649.50	\$3,954.35	\$7,048.84	\$3,483.88
82/06/30	\$7,460.00	\$4,117.03	\$7,048.84	\$3,483.88
82/07/30	\$7,460.00	\$4,117.03	\$7,048.84	\$3,483.88
82/08/31	\$9,000.00	\$4,575.72	\$9,523.84	\$4,911.12

PAGE 1

Note: Since this request simply illustrates how to identify column titles and horizontal sort values in a report, it omits a StyleSheet.

Syntax: **How to Identify a Column Title**

TYPE=TITLE, [COLUMN=*column*]

where:

COLUMN

Is used to specify one or more column titles. If you omit this attribute and value, the formatting will be applied to all of the report's column titles.

column

Specifies the column whose title you wish to format. For column values, see [How to Identify an Entire Column](#) on page 529.

Syntax: **How to Identify a Horizontal Sort Title or Value**

TYPE={ACROSSTITLE|ACROSSVALUE}, [ACROSS=*column*]

where:

ACROSSTITLE

Specifies a horizontal sort (ACROSS) title.

ACROSSVALUE

Specifies a horizontal sort (ACROSS) value.

Although horizontal sort values are not technically titles, they often function as titles that categorize the column titles appearing beneath them.

ACROSS

Is used to specify titles or values for a specific horizontal sort field. If you omit this attribute and value, the formatting will be applied to the titles or values of all of the report's horizontal sort fields.

column

Specifies the horizontal sort (ACROSS) field whose title or values you wish to format. For values you can assign to this attribute, see [How to Identify a Row of Horizontal Sort \(ACROSS\) Data](#) on page 541.

Example: Identifying Column Titles and Horizontal Sort (ACROSS) Values

The following illustrates how to identify vertical sort titles, horizontal sort titles, and horizontal sort values. The vertical sort titles (TYPE=TITLE) are Manufacturing Plant, Quantity Sold and Product Cost; the horizontal sort title (TYPE=ACROSSTITLE) is Year; and the horizontal sort values (TYPE=ACROSSVALUE) are 1999, 2000, and 2001. The StyleSheet declarations that identify these components are highlighted in the request.

```
TABLE FILE CENTORD
SUM QUANTITY AS 'Quantity,Sold' LINE_COGS/I9 AS 'Product,Cost'
BY PLANT
ACROSS YEAR
WHERE YEAR EQ '2000' OR '2001' OR '2002'
HEADING
"Plant Production Cost Analysis"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
TYPE=TITLE, STYLE=BOLD, $
TYPE=ACROSSTITLE, STYLE=BOLD, $
TYPE=ACROSSVALUE, STYLE=BOLD+ITALIC, COLOR=BLUE, $
TYPE=ACROSSVALUE, COLUMN=N4, STYLE=BOLD, COLOR=RED, $
ENDSTYLE
END
```

The output is:

Plant Production Cost Analysis						
YEAR						
	2000		2001		2002	
Manufacturing Plant	Quantity Sold	Product Cost	Quantity Sold	Product Cost	Quantity Sold	Product Cost
BOS	3,902	955189	491,080	109839382	538,836	120518527
DAL	1,122	230249	185,785	41129581	203,937	45147907
LA	5	986	109,326	24027600	119,925	26356066
ORL	.	.	184,519	41143292	202,390	45127226
SEA	.	.	41,331	9578621	45,349	10510177
STL	2,019	458317	369,456	81807263	405,268	89734521

Syntax: How to Identify Free Text in an FML Report

`TYPE=FREETEXT, LABEL={Rn|label}`

where:

Rn

Is an implicit row label. To determine the value of *n*, count the number of rows up to and including the desired row.

label

Is an explicit row label.

Example: Identifying Free Text in an FML Report

The following illustrates how to identify free text in an FML report. In this example, the free text are the rows "CASH ACCOUNTS" and "OTHER CURRENT ASSETS." The relevant StyleSheet declarations are highlighted:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
" --- CASH ACCOUNTS ---" LABEL CA          OVER
1010 AS 'CASH ON HAND'                     OVER
1020 AS 'DEMAND DEPOSITS'                  OVER
1030 AS 'TIME DEPOSITS'                    OVER
" "                                         OVER
" --- OTHER CURRENT ASSETS ---" LABEL OCA  OVER
1100 AS 'ACCOUNTS RECEIVABLE'              OVER
1200 AS 'INVENTORY'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=FREETEXT, LABEL=CA, STYLE=BOLD, SIZE=12, $
TYPE=FREETEXT, LABEL=OCA, STYLE=BOLD, SIZE=12, $
ENDSTYLE
END
```

The output is:

	<u>AMOUNT</u>
--- 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

Syntax: **How to Create a Custom Report Title**

Add the following declaration to your StyleSheet

```
TYPE=REPORT, TITLETEXT='title', $
```

where:

title

Is the text for your title.

The maximum amount of characters for:

- ❑ The worksheet tab name in an EXL2K report is 128. Any text that exceeds 128 characters will be truncated.
- ❑ The browser title for an HTML report is 95. This is a limit imposed by the browser.

Text specified in the title is placed in the file as is and is not encoded. Special characters, such as <, >, &, and so on, should not be used as they have special meaning in HTML and may produce unpredictable results.

Note: The words "Microsoft Internet Explorer" are always appended to any HTML report title.

For an example of using this technique, see [Working With Excel 2000 and Excel 97 Reports](#) on page 635.

Identifying a Heading or Footing

How to:

- Identify a Heading or Footing
- Identify an Individual Line in a Heading or Footing
- Identify a Text String in a Heading or Footing
- Identify an Embedded Field in a Heading or Footing
- Insert the Total Page Count
- Display the Total Number of Pages Within Each Sort Group

Within a StyleSheet, you can identify a report's headings and footings, and the individual lines, text strings, and fields within them, in order to format them.

You can use the <TABLASTPAGE system variable to insert the total number of pages in a heading or footing. For example, if you want to add a footing in your report that reads "Page 1 of 5", you can use the <TABLASTPAGE system variable in conjunction with the <TABPAGENO system variable to do so.

You can also use the <BYLASTPAGE system variable to display the number of pages of output within each sort group when a report uses the REPAGE option to reset the page numbers for each sort group. If the REPAGE option is not used in the report, the total number of pages in the report (<TABLASTPAGE variable) is used for <BYLASTPAGE.

The following example illustrates where a report heading (TABHEADING), a page heading (HEADING), a sort heading (SUBHEAD), a sort footing (SUBFOOT), and a report footing (TABFOOTING) are stored in a report, and which TYPE values you use to identify them.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL HIRE_DATE
BY LAST_NAME
BY FIRST_NAME
ON TABLE SUBHEAD
"CONFIDENTIAL INFORMATION"
"SWIFTY INFORMATION GROUP - EMPLOYEE LIST BY DEPARTMENT"
HEADING CENTER
"</1>EMPLOYEE LIST FOR DEPARTMENT: <DEPARTMENT"
ON LAST_NAME SUBHEAD
"ID: <EMP_ID"
ON LAST_NAME SUBFOOT
"*** REVIEW SALARY FOR <FIRST_NAME <LAST_NAME"
FOOTING
"CONFIDENTIAL INFORMATION"
ON TABLE SUBFOOT
"</1>***END OF REPORT***"
END
```

The output is:

```
PAGE      1

CONFIDENTIAL INFORMATION
SWIFTY INFORMATION GROUP - EMPLOYEE LIST BY DEPARTMENT  ← TYPE = TABHEADING

      EMPLOYEE LIST FOR DEPARTMENT: PRODUCTION ← TYPE = HEADING

LAST_NAME      FIRST_NAME      CURR_SAL  HIRE_DATE
-----
ID: 119329144 ← TYPE = SUBHEAD
BANNING        JOHN           $29,700.00  82/08/01
** REVIEW SALARY FOR JOHN BANNING ← TYPE = SUBFOOT

ID: 326179357 ← TYPE = SUBHEAD
BLACKWOOD     ROSEMARIE      $21,780.00  82/04/01
** REVIEW SALARY FOR ROSEMARIE BLACKWOOD
CONFIDENTIAL INFORMATION ← TYPE = FOOTING

***END OF REPORT*** ← TYPE = TABFOOTING
```

Note: Since this request simply illustrates how to identify different types of headings and footings, it omits a StyleSheet.

Syntax: How to Identify a Heading or Footing

```
TYPE=headfoot, [BY=sortcolumn]
```

where:

headfoot

Identifies a heading or footing. Select from:

TABHEADING which is a report heading. This appears once at the beginning of the report and is generated by ON TABLE SUBHEAD.

TABFOOTING which is a report footing. This appears once at the end of the report and is generated by ON TABLE SUBFOOT.

HEADING which is a page heading. This appears at the top of every report page and is generated by HEADING.

FOOTING which is a page footing. This appears at the bottom of every report page and is generated by FOOTING.

SUBHEAD which is a sort heading. This appears at the beginning of a vertical (BY) sort group (generated by ON *sortfield* SUBHEAD).

SUBFOOT which is a sort footing. This appears at the end of a vertical (BY) sort group (generated by ON *sortfield* SUBFOOT).

BY

When there are several sort headings or sort footings, each associated with a different vertical sort (BY) column, you can identify which sort heading or sort footing you wish to format.

If there are several sort headings or sort footings associated with different vertical sort (BY) columns, and you omit this attribute and value, the formatting will be applied to all of the sort headings or footings.

sortcolumn

Specifies the vertical sort (BY) column associated with one of the report's sort headings or sort footings.

Syntax: How to Identify an Individual Line in a Heading or Footing

`TYPE=type, LINE=line_#`

where:

type

Identifies a type of heading or footing. Select from HEADING, FOOTING, TABHEADING, TABFOOTING, SUBHEAD, or SUBFOOT. For details, see [How to Identify a Heading or Footing](#) on page 555.

line_#

Identifies a line by its position in the heading or footing.

Example: Identifying an Individual Line in a Heading

The following example illustrates how to format individual lines in a heading. Heading line 1 (Sales Quantity Analysis) is formatted in bold, point size 11. Heading line 2 (**Confidential**) is formatted in bold and red. The relevant StyleSheet declarations are highlighted in the request.

```
TABLE FILE CENTORD
HEADING
"Sales Quantity Analysis"
***Confidential***
" "
SUM QUANTITY
ACROSS YEAR
BY PLANT
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=HEADING, LINE=1, SIZE=11, STYLE=BOLD,$
TYPE=HEADING, LINE=2, COLOR=RED, STYLE=BOLD,$
TYPE=HEADING, JUSTIFY=CENTER, $
ENDSTYLE
END
```

The output is:

Sales Quantity Analysis			
Confidential			
	YEAR		
	2000	2001	2002
Manufacturing			
Plant			
BOS	3,902	491,080	538,836
DAL	1,122	185,785	203,937
LA	5	109,326	119,925
ORL	.	184,519	202,390
SEA	.	41,331	45,349
STL	2,019	369,456	405,268

Syntax: How to Identify a Text String in a Heading or Footing

`TYPE=type, [LINE=line_#], [OBJECT=TEXT], ITEM=item_#`

where:

type

Identifies a type of heading or footing. Select from HEADING, FOOTING, TABHEADING, TABFOOTING, SUBHEAD, or SUBFOOT. For details, see [Identifying a Heading or Footing](#) on page 554.

line_#

Identifies a line by its position in the heading or footing. You need to include the LINE attribute only if you have a multi-line heading or footing.

TEXT

Formats only text strings and Dialogue Manager variables (also known as &variables). It is not necessary to use OBJECT=TEXT in your declaration unless you are styling both text strings and embedded fields in the same heading or footing.

item_#

Identifies an item by its position in a line.

If you need to apply formatting to several parts of a continuous text string that appears on one line, you can break the header or footer into multiple parts using spot markers. Place the spot marker after the text string you wish to specify. The <+0> spot marker will not add any additional spaces to your heading or footing. When using spot markers, text is divided as follows:

```
TABLE FILE GGSales
SUM UNITS DOLLARS BY CATEGORY BY PRODUCT
HEADING
\First Quarter <+0>Sales <+0>Report"
ON TABLE SET STYLE *
TYPE=HEADING, OBJECT=TEXT, ITEM=1, FONT=ARIAL, $
TYPE=HEADING, OBJECT=TEXT, ITEM=2, SIZE=14, $
TYPE=HEADING, OBJECT=TEXT, ITEM=3, STYLE=BOLD, $
ENDSTYLE
END
```

For an example, see [How to Identify a Text String in a Heading or Footing](#) on page 558.

The position value also depends on whether you are using the OBJECT attribute or not. If you are using:

- OBJECT=TEXT, count only text strings from left to right.
- No OBJECT, count text strings and embedded field values from left to right.

Example: Identifying a Text String in a Heading Using Spot Markers

The following illustrates how to apply different formats to text strings in a heading using spot markers. The spot markers used in this example are <+0>, since they do not add any spaces. The relevant StyleSheet declarations are highlighted in the request.

```
TABLE FILE CENTORD
HEADING
"Third Quarter,<+0>2002:<+0> Sales Quantity Analysis"
SUM QUANTITY BY PLANT
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=HEADING, OBJECT=TEXT, ITEM=1, STYLE=BOLD+UNDERLINE, SIZE=12, $
TYPE=HEADING, OBJECT=TEXT, ITEM=2, COLOR=BLUE, SIZE=12,
    STYLE=BOLD+UNDERLINE, $
TYPE=HEADING, OBJECT=TEXT, ITEM=3, STYLE=ITALIC,$
ENDSTYLE
END
```

The output is:

Third Quarter,2002: *Sales Quantity Analysis*

Manufacturing

<u>Plant</u>	<u>Quantity:</u>
BOS	1,033,818
DAL	390,844
LA	229,256
ORL	386,909
SEA	86,680
STL	776,743

Syntax: How to Identify an Embedded Field in a Heading or Footing

```
TYPE=type, [LINE=line_#], OBJECT=FIELD, [ITEM=item #]
```

where:

type

Identifies a type of heading or footing. Select from HEADING, FOOTING, TABHEADING, TABFOOTING, SUBHEAD, or SUBFOOT. For details, see *Identifying a Heading or Footing* on page 554.

line_#

Identifies a line by its position in the heading or footing. You need to include the LINE attribute only if you have a multi-line heading or footing.

item_#

Identifies an item by its position in a line.

If you have more than one embedded field in a heading or footing, you must specify the field you wish to format by giving the item number. Count items from left to right. Do not include text fields in the count. You do not need to specify the item number if there is only one embedded field in the heading or footing.

Example: Identifying Embedded Fields in a Heading

The following illustrates how to format an embedded field in a heading. Notice that the item number is not specified in the StyleSheet declaration since there is only one embedded field in the heading. The relevant StyleSheet declaration is highlighted in the request.

```
TABLE FILE CENTORD
HEADING
"Sales For <YEAR By Plant"
SUM QUANTITY BY PLANT
WHERE YEAR EQ 2000
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=HEADING, OBJECT=TEXT, COLOR=BLUE,$
TYPE=HEADING, OBJECT=FIELD, COLOR=RED, STYLE=BOLD,$
ENDSTYLE
END
```

The output is:

Sales For **2000** By Plant

Manufacturing

<u>Plant</u>	<u>Quantity</u>
BOS	3,902
DAL	1,122
LA	5
STL	2,019

Syntax: How to Insert the Total Page Count

To insert the total number of pages, add the following to your request:

```
<TABLASTPAGE
```

Note that TABLASTPAGE is not supported with Excel 2000.

Example: Inserting the Current Page Number and the Total Page Count

The following illustrates how to add the current page number and the total page count to a report. The relevant syntax is highlighted in the request.

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AS 'Employee ID'
BY SALARY IN-GROUPS-OF 5000 AS 'Salary'
BY PCT_INC AS 'Percent,Increase'
BY DAT_INC AS 'Date of,Increase'
ON SALARY PAGE-BREAK
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=TITLE, STYLE=BOLD, SIZE=11, $
ENDSTYLE
FOOTING
"Page <TABPAGENO of <TABLASTPAGE"
END
```

The first two pages of output are:

<u>Salary</u>	<u>Percent Increase</u>	<u>Date of Increase</u>	<u>Employee ID</u>
\$5,000.00	.00	82/01/04	119265415
		82/04/01	543729165
	.04	82/06/11	543729165
	.05	82/05/14	119265415

Page 1 of 5

<u>Salary</u>	<u>Percent Increase</u>	<u>Date of Increase</u>	<u>Employee ID</u>
\$10,000.00	.10	82/01/01	071382660
			112847612
	.12	81/01/01	071382660

Page 2 of 5

Syntax: **How to Display the Total Number of Pages Within Each Sort Group**

The request must have the following syntax and hold the output in a styled output format:

```
BY sortfield REPAGE
```

The heading or footing can use the following syntax to display "Page x of y"

```
{HEADING|FOOTING}  
"Page <TABPAGENO of <BYLASTPAGE"
```

where:

sortfield

Is the sort field that has the REPAGE option. A PAGE-BREAK is required on the same sort field or a lower level sort field. PAGE-BREAK starts a new page for each sort break. REPAGE resets the page number to 1 for each sort break.

<TABPAGENO

Is the current page number.

<BYLASTPAGE

Is the last page number before the repage.

Example: Paginating Within a Sort Group

The following request against the GGSALES data source sorts by product, region, category, and city. It resets the pagination each time the product changes. The heading prints the current page number and the total within each product group:

```
TABLE FILE GGSALES
HEADING CENTER
"<PRODUCT : Page <TABPAGENO of <BYLASTPAGE "

SUM UNITS
BY PRODUCT NOPRINT REPAGE
BY REGION PAGE-BREAK
BY CATEGORY
BY CITY

ON TABLE HOLD FORMAT PDF
END
```

The following partial output shows that the page number resets to 1 when the product changes and that the BYLASTPAGE variable displays the total number of pages for each product:

Biscotti : Page 1 of 4			
<u>Region</u>	<u>Category</u>	<u>City</u>	<u>Unit Sales</u>
Midwest	Food	Chicago	29413
		Houston	27504
		St. Louis	29188

Biscotti : Page 2 of 4			
<u>Region</u>	<u>Category</u>	<u>City</u>	<u>Unit Sales</u>
Northeast	Food	Boston	47064
		New Haven	46214
		New York	51964

Biscotti : Page 3 of 4

<u>Region</u>	<u>Category</u>	<u>City</u>	<u>Unit Sales</u>
Southeast	Food	Atlanta	43639
		Memphis	35349
		Orlando	40606

Biscotti : Page 4 of 4

<u>Region</u>	<u>Category</u>	<u>City</u>	<u>Unit Sales</u>
West	Food	Los Angeles	20773
		San Francisco	22987
		Seattle	26676

Capuccino : Page 1 of 3

<u>Region</u>	<u>Category</u>	<u>City</u>	<u>Unit Sales</u>
Northeast	Coffee	Boston	15358
		New Haven	12386
		New York	17041

Identifying a Page Number, Underline, or Skipped Line

How to:

Identify a Page Number, Underline, or Skipped Line

Extend an Underline to the Entire Report Column

Format a Blank Line

Format an Underline

Add or Remove a Report Component Underline

Remove an Underline From a Column Title

Reference:

Usage Notes for the EXTUNDERLINE Attribute

Section Separation Features

In a report, you can identify and format page numbers, underlines, and skipped lines using the PAGENUM, SKIPLINE, and UNDERLINE attributes.

You can make a detailed tabular report easier to read by separating sections with blank lines or underlines.

You cannot add blank lines or underlines to an HTML report that displays a grid. You can add blank lines or underlines if you set the GRID attribute to OFF.

When inserting blank lines, the setting of the LINES parameter should be at least one less than the setting of the PAPER parameter, to allow room for blanks after the display of data on a page.

Note that although you can insert skipped lines and underlines in an HTML report, formatting is not supported.

The following illustrates where the PAGENUM, UNDERLINE, and SKIPLINE components appear in a report, and which TYPE values you use to identify them.

```
TABLE FILE CENTORD
HEADING
"Sales By Plant"
SUM QUANTITY
BY PLANT BY YEAR
WHERE PLANT EQ 'BOS' OR 'DAL'
ON YEAR UNDER-LINE
ON PLANT SKIP-LINE
ON TABLE HOLD FORMAT PDF
END
```

TYPE=PAGENUM →	PAGE	1		
	Sales By Plant			
	<u>Manufacturing Plant</u>	<u>Year</u> <u>Quantity</u>		
TYPE=UNDERLINE →	}	BOS	1999	262,167
			2000	438,155
			2001	479,319
			2002	491,080
TYPE=SKIPLINE →		DAL	1999	94,611
			2000	162,290
			2001	171,346
			2002	184,385

Note: Since this request simply illustrates where the components appear in a report, it omits a StyleSheet.

Syntax: **How to Identify a Page Number, Underline, or Skipped Line**

```
TYPE=type
```

where:

```
type
```

Identifies the report component. Select from:

PAGENUM which identifies page numbers. Note that the TABPAGENO variable is a component of the heading or footing in which it appears and can be formatted as a subcomponent of that heading or footing.

SKIPLINE which denotes skipped lines generated by ON *field* SKIP-LINE. This is not supported for reports in HTML format.

UNDERLINE which identifies underlines generated by ON *field* UNDER-LINE, or by BAR in a Financial Modeling Language (FML) report. This is not supported for reports in HTML format.

Example: Identifying Underlines and Page Numbers

The following illustrates how to identify underlines and page numbers in a report request. The relevant StyleSheet declarations appear in boldface in the request.

Note that this report is formatted in PDF, since formatting is not supported for underlines in an HTML report.

```
TABLE FILE CENTORD
HEADING
"Sales By Plant"
SUM QUANTITY
BY PLANT BY YEAR
WHERE PLANT EQ 'BOS' OR 'DAL' OR 'LA'
ON PLANT UNDER-LINE SKIP-LINE
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=HEADING, OBJECT=TEXT, COLOR=BLUE, FONT=ARIAL,$
TYPE=PAGENUM, STYLE=ITALIC, SIZE=8,$
TYPE=UNDERLINE, COLOR=RED,$
ENDSTYLE
END
```

The output is:

PAGE 1

Sales By Plant

Manufacturing
Plant

<u>Plant</u>	<u>YEAR</u>	<u>Quantity:</u>
BOS	2000	3,902
	2001	491,080
	2002	538,836
DAL	2000	1,122
	2001	185,785
	2002	203,937
LA	2000	5
	2001	109,326
	2002	119,925

Example: Identifying Skipped Lines

The following illustrates how to identify skipped lines in a report. The relevant StyleSheet declaration is highlighted in the request.

```
TABLE FILE CENTINV
HEADING
"Low Stock Report"
" "
SUM QTY_IN_STOCK
WHERE QTY_IN_STOCK LT 5000
BY PRODNAME
ON PRODNAME SKIP-LINE
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=SKIPLINE, BACKCOLOR=SILVER, $
ENDSTYLE
END
```

The output is:

Low Stock Report

<u>Product Name:</u>	<u>Quantity In Stock:</u>
DVD Upgrade Unit for Cent. VCR	199
R5 Micro Digital Tape Recorder	1990
110 VHS-C Camcorder 20 X	4000
120 VHS-C Camcorder 40 X	2300
340SX Digital Camera 65K P	990
650DL Digital Camcorder 150 X	2972

Syntax: How to Extend an Underline to the Entire Report Column

By default, underlines for column titles on a report extend only from the beginning to the end of the column title text. You can extend the underline to the entire report column in styled report output using the EXTUNDERLINE option in your WebFOCUS StyleSheet. EXTUNDERLINE is an option of the STYLE attribute for the TITLE report component. It is supported for formats DHTML, PDF, PS, and PPT.

```
TYPE = TITLE, [COLUMN = colspec,] STYLE = [+|-]EXTUNDERLINE , $
```

where:

colspec

Is any valid column specification.

+EXTUNDERLINE

Adds the EXTUNDERLINE option to the inherited text style or specifies a combination of text styles (for example, STYLE=BOLD+UNDERLINE).

-EXTUNDERLINE

Removes the EXTUNDERLINE option from the inherited text style.

Reference: Usage Notes for the EXTUNDERLINE Attribute

- ❑ HTML format is not supported because the browser calculates the column width and renders the report.
- ❑ GRID=ON and EXTUNDERLINE are mutually exclusive since the GRID line spans the width of the column. GRID overrides any styling specified for the column title underline.

Example: Extending an Underline to the Entire Report Column

The following request against the GGSALES data source sums dollar sales by city and by date:

```
DEFINE FILE GGSALES
YEAR/YY = DATE;
MONTH/M = DATE;
END
TABLE FILE GGSALES
SUM DOLLARS
BY DATE
BY CITY
WHERE YEAR EQ 1997
WHERE MONTH FROM 01 TO 05
WHERE CITY EQ 'Seattle' OR 'San Francisco' OR 'Los Angeles'
ON TABLE SET PAGE NOPAGE
ON TABLE HOLD FORMAT DHTML
END
```

The output shows that only the column titles are underlined:

<u>Date</u>	<u>City</u>	<u>Sales</u>
1997/01/01	Los Angeles	159935
	San Francisco	162712
	Seattle	159804
1997/02/01	Los Angeles	148915
	San Francisco	143987
	Seattle	132505
1997/03/01	Los Angeles	165902
	San Francisco	145129
	Seattle	165847
1997/04/01	Los Angeles	146106
	San Francisco	158799
	Seattle	144169
1997/05/01	Los Angeles	178336
	San Francisco	144534
	Seattle	190208

To underline entire columns, generate the output in a format that can be styled and use the EXTUNDERLINE option in the STYLE attribute for the TITLE component. For example, the following request creates DHTML output in which the column titles are in boldface and left justified, and the underline is extended to the entire report columns:

```

DEFINE FILE GGSALES
YEAR/YY = DATE;
MONTH/M = DATE;
END
TABLE FILE GGSALES
SUM DOLLARS
BY DATE
BY CITY
WHERE YEAR EQ 1997
WHERE MONTH FROM 01 TO 05
WHERE CITY EQ 'Seattle' OR 'San Francisco' OR 'Los Angeles'
ON TABLE SET PAGE NOPAGE
ON TABLE HOLD FORMAT DHTML
ON TABLE SET STYLE *
TYPE=TITLE, STYLE= BOLD +EXTUNDERLINE, JUSTIFY=LEFT $
ENDSTYLE
END

```

The output is:

<u>Date</u>	<u>City</u>	<u>Sales</u>
1997/01/01	Los Angeles	159935
	San Francisco	162712
	Seattle	159804
1997/02/01	Los Angeles	148915
	San Francisco	143987
	Seattle	132505
1997/03/01	Los Angeles	165902
	San Francisco	145129
	Seattle	165847
1997/04/01	Los Angeles	146106
	San Francisco	158799
	Seattle	144169
1997/05/01	Los Angeles	178336
	San Francisco	144534
	Seattle	190208

The following version of the request makes the EXTUNDERLINE and JUSTIFY=LEFT options the default for the TITLE component, then makes the Date column title bold and removes the extended underline from that column:

```

DEFINE FILE GGSALES
YEAR/YY = DATE;
MONTH/M = DATE;
END
TABLE FILE GGSALES
SUM DOLLARS AS 'Sales'
BY DATE
BY CITY
WHERE YEAR EQ 1997
WHERE MONTH FROM 01 TO 05
WHERE CITY EQ 'Seattle' OR 'San Francisco' OR 'Los Angeles'
ON TABLE SET PAGE NOPAGE
ON TABLE HOLD FORMAT DHTML
ON TABLE SET STYLE *
TYPE=TITLE,STYLE= EXTUNDERLINE, JUSTIFY=LEFT , $
TYPE=TITLE,COLUMN= DATE, STYLE= -EXTUNDERLINE +BOLD , $
ENDSTYLE
END

```

The output is:

<u>Date</u>	<u>City</u>	<u>Sales</u>
1997/01/01	Los Angeles	159935
	San Francisco	162712
	Seattle	159804
1997/02/01	Los Angeles	148915
	San Francisco	143987
	Seattle	132505
1997/03/01	Los Angeles	165902
	San Francisco	145129
	Seattle	165847
1997/04/01	Los Angeles	146106
	San Francisco	158799
	Seattle	144169
1997/05/01	Los Angeles	178336
	San Francisco	144534
	Seattle	190208

Reference: Section Separation Features

Feature	Description	Applies to
<code>SKIP-LINE</code>	Adds a blank line.	HTML (requires GRID=OFF) PDF PS
<code>TYPE=SKIPLINE</code>	Formats a blank line.	PDF PS
<code>UNDER-LINE</code>	Underlines a sort group.	HTML (requires GRID=OFF) PDF PS
<code>TYPE=UNDERLINE</code>	Formats an underline.	PDF PS

Feature	Description	Applies to
<code>STYLE={+ -}UNDERLINE</code>	Adds an underline to a report component, or removes an underline from a report component other than a column title.	HTML PDF PS
<code>BAR AS '{- =}'</code>	Selects a light or heavy underline in an FML report.	HTML PDF (displays single or double underline)

Syntax: **How to Format a Blank Line**

`TYPE=SKIPLINE, attribute=value, $`

where:

attribute

Is a valid StyleSheet attribute.

value

Is the value of the attribute.

Note: This option is supported for PDF, PS, and HTML reports (when used in conjunction with internal Cascading Style Sheets).

Syntax: **How to Format an Underline**

`TYPE=UNDERLINE ... COLOR={color|RGB} (r g b), $`

where:

UNDERLINE

Denotes underlines generated by ON *fieldname* UNDER-LINE.

COLOR

Specifies the color of the underline. If the display or output device does not support colors, it substitutes shades of gray. The default value is black.

color

Is one of the supported color values.

RGB

Specifies the text color using a mixture of red, green, and blue.

(r g b)

Is the desired intensity of red, green, and blue, respectively. The values are on a scale of 0 to 255, where 0 is the least intense and 255 is the most intense.

Note that using the three-color components in equal intensities results in shades of gray. For more information, see [Color Values in a Report](#) on page 522.

Note: This option is supported for PDF, PS, and HTML reports (when used in conjunction with internal Cascading Style Sheets).

Example: Formatting a Sort Group Underline

This request uses UNDERLINE to change the default color of an underline from black to red.

```
TABLE FILE GGSALES
SUM UNITS DOLLARS
BY CATEGORY BY PRODUCT
HEADING
"Sales Report"
" "
ON CATEGORY UNDER-LINE
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=UNDERLINE, COLOR=RED, $
ENDSTYLE
END
```

The result is an eye-catching separation between sort group values. The online PDF report appears as:

Sales Report

<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
Coffee	Capuccino	189217	2381590
	Espresso	308986	3906243
	Latte	878063	10943622
Food	Biscotti	421377	5263317
	Croissant	630054	7749902
	Scone	333414	4216114
Gifts	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829

Syntax: **How to Add or Remove a Report Component Underline**

TYPE=*type*, [*subtype*,] STYLE=[+|-]UNDERLINE, \$

where:

type

Is the report component.

subtype

Are additional attributes, such as COLUMN, ACROSS, or ITEM, needed to identify the report component.

±

Adds an underline to the inherited text style or specifies a combination of text styles (for example, STYLE=BOLD+UNDERLINE). ± is the default value.

-

Removes an underline from an inherited text style.

For more information, see [Identifying Report Components](#) on page 525.

Syntax: How to Remove an Underline From a Column Title

This syntax applies to an HTML report with internal Cascading Style Sheet.

```
TYPE=TITLE, [COLUMN=column,] STYLE=-UNDERLINE, $
```

where:

```
COLUMN=column
```

Specifies a column.

Example: Adding Column Underlines and Removing Column Title Underlines

This request adds underlines to the values of the column CATEGORY and removes the default underlines from the column titles in an HTML report with an internal Cascading Style Sheet.

```
SET HTMLCSS = ON
TABLE FILE MOVIES
PRINT TITLE DIRECTOR
BY CATEGORY
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=TITLE, STYLE=-UNDERLINE, $
TYPE=REPORT, COLUMN=CATEGORY, STYLE=UNDERLINE, $
ENDSTYLE
END
```

The partial report is:

CATEGORY	TITLE	DIRECTOR
<u>ACTION</u>	JAWS	SPIELBERG S.
	ROBOCOP	VERHOVEN P.
	TOTAL RECALL	VERHOVEN P.
	TOP GUN	SCOTT T.
	RAMBO III	MCDONALD P.
<u>CHILDREN</u>	SMURFS, THE	
	SHAGGY DOG, THE	BARTON C.
	SCOOBY-DOO-A DOG IN THE RUFF	
	ALICE IN WONDERLAND	GEROMINI
	SESAME STREET-BEDTIME STORIES AND SONGS	
	ROMPER ROOM-ASK MISS MOLLY	
	SLEEPING BEAUTY	DISNEY W.
<u>CLASSIC</u>	BAMBI	DISNEY W.
	EAST OF EDEN	KAZAN E.
	CITIZEN KANE	WELLES O.
	CYRANO DE BERGERAC	GORDON M.
	MARTY	MANN D.
	MALTESE FALCON, THE	HUSTON J.
	GONE WITH THE WIND	FLEMING V.

Reusing FOCUS StyleSheet Declarations With Macros

In this section:

Defining a FOCUS StyleSheet Macro

Applying a FOCUS StyleSheet Macro

If you frequently use a group of attributes within a StyleSheet declaration, you can create a StyleSheet macro that groups the sequence of attributes together, enabling you to apply them repeatedly throughout the StyleSheet without recoding them.

Defining a FOCUS StyleSheet Macro

How to:

Define a FOCUS StyleSheet Macro

A StyleSheet macro must be defined in the StyleSheet that references it, and the macro definition must precede its use in the StyleSheet.

To define a macro, use the DEFMACRO attribute followed by the desired styling attributes.

Syntax: How to Define a FOCUS StyleSheet Macro

```
DEFMACRO = macroname, attribute1 = value1, [attribute2 = value2,]... $
```

where:

macroname

Is the name you assign to the macro you are creating.

attribute

Is any StyleSheet attribute, such as an attribute to format a report component, insert a graphic, or apply a condition for conditional formatting (WHEN).

value

Is the value you want to assign to the attribute.

Applying a FOCUS StyleSheet Macro

How to:

Apply a FOCUS StyleSheet Macro

A StyleSheet macro applies all the formatting defined in the macro to the report component specified in the declaration. To apply a macro, use the MACRO attribute. You can apply one macro per declaration.

When applying a StyleSheet macro to a report component, you can override any attribute defined in the macro by specifying the same attribute with the new value in that declaration, following the MACRO attribute..

Syntax: **How to Apply a FOCUS StyleSheet Macro**

`TYPE=type, [subtype,] MACRO=macroname, [condition,] $`

where:

type

Is the report component you wish to affect.

subtype

Is any additional attribute, such as COLUMN, ACROSS, or ITEM, that is needed to identify the report component to which you are applying the macro. See [Identifying Report Components](#) on page 525, for information about how to specify different types of report components.

macroname

Is the name of the macro to apply to the specified report component. The macro must be defined in the same StyleSheet.

condition

Is an optional WHEN attribute that you can specify to make this declaration conditional.

Example: **Defining, Applying, and Overriding a FOCUS StyleSheet Macro**

The following annotated example illustrates how to define, apply, and override macros in your StyleSheet:

```
TABLE FILE GGSales
  SUM UNITS DOLLARS
  BY CATEGORY BY PRODUCT
  HEADING
  "Sales Report"
  FOOTING
  "Sales Report - Page <TABPAGENO"
  ON TABLE HOLD FORMAT HTML
  ON TABLE SET STYLE *
  TYPE=REPORT, GRID=OFF,$
1.  DEFMACRO=A, STYLE=BOLD, SIZE=12, $
2.  DEFMACRO=BI, STYLE=BOLD+ITALIC, COLOR=PURPLE, $
3.  TYPE=HEADING, MACRO=A, $
4.  TYPE=FOOTING, MACRO=BI, COLOR=BLACK, $
5.  TYPE=DATA, COLUMN=N1, MACRO=BI, $
  ENDSTYLE
END
```

- 1.** Defines the A macro.
- 2.** Defines the BI macro.

3. Illustrates how the A macro is applied to the heading.
4. Illustrates how the BI macro is applied to the footing and is partially overridden by the attribute value pair COLOR=BLACK.
5. Illustrates how the BI macro is applied to the data in the BY sort field CATEGORY (specified by TYPE=DATA, COLUMN=N1).

The output is:

Sales Report

<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
<i>Coffee</i>	Capuccino	189217	2381590
	Espresso	308986	3906243
	Latte	878063	10943622
<i>Food</i>	Biscotti	421377	5263317
	Croissant	630054	7749902
	Scone	333414	4216114
<i>Gifts</i>	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829

Sales Report - Page 1

FOCUS StyleSheet Attribute Inheritance

Reference:

FOCUS StyleSheet Inheritance Hierarchy

Each report component inherits StyleSheet attributes from its parent component. You can override an inherited attribute by explicitly specifying the same attribute with a different value in the child component's declaration. Since each component inherits automatically, you need specify only those attributes that differ from, or that augment, a component's inherited attributes.

Inheritance enables you to define common formatting in a single declaration, and to apply it automatically to all child components, except for those components for which you specify different attribute values to override the inherited values. You benefit from less coding and a more concise StyleSheet.

For example, you could specify that all report titles should be blue and bold:

```
TYPE=TITLE, COLOR=BLUE, STYLE=BOLD, $
```

Each column title will inherit this formatting, appearing in blue and bold by default. However, you can choose to format one column differently, allowing it to inherit the blue color, but specifying that it override the bold style and that it add a yellow background color:

```
TYPE=TITLE, COLUMN=N2, STYLE=-BOLD, BACKCOLOR=YELLOW, $
```

Reference: FOCUS StyleSheet Inheritance Hierarchy

Report components inherit StyleSheet attributes according to a hierarchy. The root of the hierarchy is the entire report, specified in a StyleSheet declaration by `TYPE=REPORT`. (Declarations that omit `TYPE` default to `TYPE=REPORT`, and so are also applied to the entire report.) Attributes that are unspecified for the entire report default to values that are determined according to the report's display format, such as HTML or PDF.

Each report component inherits from its parent component. Component X is a parent of component Y if X is specified by a subset of all the "type" attributes that specify Y, and if those shared type attributes have the same values. For example,

- ❑ A component specified by `TYPE=x, subtype=y, elementtype=z` is a child of the component specified by `TYPE=x, subtype=y` and inherits attributes from it.
- ❑ The component specified by `TYPE=x, subtype=y` is a child of the component specified by `TYPE=x`, and inherits from it.
- ❑ The component specified by `TYPE=x`, where `x` is any value other than `REPORT`, is a child of the entire report (`TYPE=REPORT`) and inherits from it.

When you use an external Cascading Style Sheet (CSS), a report component inherits formatting from parent HTML elements, not from a parent report component.

Example: Augmenting Inherited FOCUS StyleSheet Attributes

The following illustrates how to augment inherited StyleSheet attributes. The StyleSheet declarations discussed in this example are highlighted in the report request.

The page heading in this report has two lines. The first StyleSheet declaration identifies the report component `HEADING` to be formatted in bold and have 12-point font size. This will format both lines of the heading with these styles.

To augment the format for the second line of the heading, a second declaration has been added that specifies the heading line number and the additional style characteristic. In this case we have added the declaration `TYPE=HEADING, LINE=2, STYLE=ITALIC`. The second line of the heading will inherit the bold style and 12-point font size from the first `HEADING` declaration, and will also receive the italic style defined in the second declaration.

```
TABLE FILE GGSALES
SUM UNITS DOLLARS
BY CATEGORY BY PRODUCT
HEADING
"Sales Report:"
"First Quarter"
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=HEADING, STYLE=BOLD, SIZE=12, $
TYPE=HEADING, LINE=2, STYLE=ITALIC, $ENDSTYLE
END
```

The output is:

Sales Report:

First Quarter

<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
Coffee	Capuccino	189217	2381590
	Espresso	308986	3906243
	Latte	878063	10943622
Food	Biscotti	421377	5263317
	Croissant	630054	7749902
	Scone	333414	4216114
Gifts	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829

Example: Overriding Inherited FOCUS StyleSheet Attributes

The following illustrates how to override StyleSheet inheritance. The StyleSheet declarations discussed in this example are highlighted in the report request.

```

TABLE FILE GGSALLES
HEADING
"Sales Report"
SUM UNITS DOLLARS
BY CATEGORY BY PRODUCT BY DATE NOPRINT
WHERE DATE GE 19960101 AND DATE LE 19960401
ON TABLE SET STYLEMODE PAGED
ON TABLE SET LINES 20
ON TABLE HOLD FORMAT HTML
FOOTING
"Page <TABPAGENO of <TABLASTPAGE"
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
1. TYPE=REPORT, BACKCOLOR=BLUE, COLOR=WHITE, $
2. TYPE=HEADING, BACKCOLOR=WHITE, COLOR=BLACK, STYLE=BOLD, SIZE=12, $
3. TYPE=FOOTING, SIZE=11, STYLE=BOLD+ITALIC, BACKCOLOR=WHITE,
   COLOR=BLACK, $
4. TYPE=FOOTING, OBJECT=FIELD, ITEM=1, STYLE=-ITALIC, $
ENDSTYLE
END

```

1. Formats the entire report (all components) to appear with a blue background and white font.
2. Overrides the inherited format for the page heading (defined in the TYPE=REPORT declaration) by specifying the background color as white and the font as black.
3. Formats the page footing as font size 11 with a bold and italic style, and overrides the report color by specifying BACKCOLOR=WHITE and COLOR=BLACK.
4. Since the <TABPAGENO system variable is part of the page footing, it inherits all of the formatting specified in the first TYPE=FOOTING declaration. This declaration overrides the inherited format for the page footing by specifying OBJECT=FIELD, ITEM=1, and removing the italic style (STYLE=-ITALIC). Note that ITEM=1 needs to be specified, since there are two embedded fields in the footing.

The output is:

Sales Report

Category	Product	Unit Sales	Dollar Sales
Coffee	Capuccino	5507	71728
		5219	67429
		6343	87290
		4170	58692
	Espresso	14499	182900
		14015	169023
		14871	184474
		11613	156396
	Latte	31469	401873
		38725	490645
		42717	520948
		36374	468137

Page 1 of 4

Conditionally Formatting in a StyleSheet

In this section:

- Applying Sequential Conditional Formatting
- Using Conditional Grid Formatting in a Field

You can conditionally format report components or display a graphic in your report based on the values in your report. Using conditional styling, you can:

- Draw attention to particular items in the report.
- Emphasize differences between significant values.
- Customize the resources to which an end user navigates from different parts of the report.

To conditionally format reports, add the WHEN attribute to a StyleSheet declaration. The WHEN attribute specifies a condition that is evaluated for each instance of a report component (that is, for each cell of a tabular report column or each free-form report page). The StyleSheet declaration is applied to each instance that satisfies the condition, and is ignored by each instance that fails to satisfy the condition.

You can also apply sequential conditional formatting.

Applying Sequential Conditional Formatting

In this section:

Using WHEN With ACROSSCOLUMN

How to:

Conditionally Format in a StyleSheet

You can apply sequential conditional logic to a report component by creating a series of declarations, each with a different condition. This is the StyleSheet equivalent of a sequence of nested IF-THEN-ELSE statements. When several conditional declarations specify the same report component (for example, the same column) and evaluate the same field in the condition, they are processed together as a group. For each instance of the report component (for example, for each cell of a column):

- 1.** The conditional declarations in the "group" are evaluated, in the order in which they are found in the StyleSheet, until one of the conditions is satisfied. That declaration is then applied to that instance of the report component. The other conditional declarations in the "group," and any non-conditional declarations that specify the same report component and the same attributes, are ignored for that instance.
- 2.** If, however, none of the conditional declarations have been satisfied for that instance, then the first unconditional declaration for that report component that specifies the same attribute(s) is applied to that instance.
- 3.** Any unconditional declarations for that report component that specify other attributes—that is, attributes that have not already been applied to the instance in Steps 1 or 2—are now applied to the instance.
- 4.** The entire process is repeated for the next instance of the report component (for example, for the next cell of the column).

Syntax: How to Conditionally Format in a StyleSheet

`TYPE=type, [subtype,] attributes, WHEN=field1 operator {field2|value}, $`
OR

`TYPE=type, [subtype,] attributes, WHEN=FORECAST, $`

where:

type

Is the value of the TYPE attribute. You can specify any report component.

subtype

Are any additional attributes, such as COLUMN, ACROSS, or ITEM, that are needed to identify the report component to which you are applying the declaration.

attributes

Are the attributes in the StyleSheet declaration that are made conditional by the WHEN attribute. They can include most formatting or graphic image attributes.

field1, field2

Identifies the report fields that are being compared. Each one can be:

- The name of a display field or vertical sort field in a tabular report. You cannot specify a horizontal sort field (ACROSS).
- A column reference in a report.
- The name of an embedded field in the heading or footing of a free-form report.

If you wish to use a field that you do not want to display in the report, you can specify the field in the report request, and use the NOPRINT option to prevent the field from being displayed (for example, PRINT *fieldname* NOPRINT).

To apply a prefix operator to a field in a report, you can:

- Use the same prefix operator in the WHEN attribute. You must refer to the field by name in the WHEN attribute (for example, WHEN=AVE.PRICE GT 300).
- Refer to the field in the WHEN attribute by column position and omit the prefix operator (for example, WHEN=N3 GT 300). This is not supported for the ST. and CT. prefix operators.

The field cannot be a packed (P) numeric field.

operator

Defines how the condition is satisfied. You can use these relational operators:

EQ where the condition is satisfied if the values on the left and right are equal. If the values being compared are alphanumeric, their case (uppercase, lowercase, or mixed case) must match.

NE where the condition is satisfied if the values on the left and right are not equal.

LT where the condition is satisfied if the value on the left is less than the value on the right.

LE where the condition is satisfied if the value on the left is less than or equal to the value on the right.

GT where the condition is satisfied if the value on the left is greater than the value on the right.

GE where the condition is satisfied if the value on the left is greater than or equal to the value on the right.

value

Is a constant, such as a number, character string, date, or date-time. You must enclose non-numeric constants, such as character strings and dates, in single quotation marks.

Although you cannot use functions or operators here to specify the value, you can define a temporary field (COMPUTE or DEFINE) using functions and operators, use the temporary field in the report, and specify it here instead of a constant.

FORECAST

Identifies fields that are generated using the FORECAST command.

Example: Applying Basic Conditional Formatting

This example illustrates how to apply conditional formatting to a report. The conditional formatting draws attention to orders that total more than \$200,000.

Notice that because a particular column is not specified in the declaration, the formatting is applied to the entire row.

```
TABLE FILE CENTORD
HEADING
"Order Revenue"
" "
SUM ORDER_DATE LINEPRICE AS 'Order,Total:'
BY HIGHEST 10 ORDER_NUM
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=REPORT, GRID=OFF, $
TYPE=DATA, BACKCOLOR=AQUA, STYLE=BOLD, WHEN=LINEPRICE GT 200000, $
TYPE=HEADING, FONT='Arial', STYLE=BOLD, SIZE=11, $
ENDSTYLE
END
```

The output is:

Order Revenue

<u>Order</u> <u>Number:</u>	<u>Date</u> <u>Of Order:</u>	<u>Order</u> <u>Total:</u>
94710	2001/01/02	\$406,964.24
94680	2001/01/02	\$421,916.60
94670	2001/01/02	\$513,868.76
94550	2001/01/02	\$496,323.64
94530	2001/01/02	\$3,472.41
94520	2001/01/02	\$261,808.72
94490	2000/12/31	\$633,723.06
94460	2001/01/02	\$3,872.39
94430	2001/01/02	\$3,033.38
94410	2001/01/02	\$2,337.28

Example: Applying Conditional Formatting to a Column

This example illustrates how you can use conditional formatting to draw attention to columns that are not specified in the condition. The WHEN condition states that the order number for orders exceeding \$200,000 should display in boldface with an aqua background.

Notice that the column that is evaluated in the WHEN condition (LINEPRICE) is different from the column that is formatted (ORDER_NUM); they do not need to be the same.

```
TABLE FILE CENTORD
HEADING
"Order Revenue"
" "
SUM ORDER_DATE LINEPRICE AS 'Order,Total:'
BY HIGHEST 10 ORDER_NUM
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE SHEET *
TYPE=REPORT, GRID=OFF, $
TYPE=DATA, COLUMN=ORDER_NUM,
    BACKCOLOR=AQUA, STYLE=BOLD, WHEN=LINEPRICE GT 200000, $
TYPE=HEADING, FONT='Arial', STYLE=BOLD, SIZE=11, $
ENDSTYLE
END
```

The output is:

Order Revenue

Order Number:	Date Of Order:	Order Total:
94710	2001/01/02	\$406,964.24
94680	2001/01/02	\$421,916.60
94670	2001/01/02	\$513,868.76
94550	2001/01/02	\$496,323.64
94530	2001/01/02	\$3,472.41
94520	2001/01/02	\$261,808.72
94490	2000/12/31	\$633,723.06
94460	2001/01/02	\$3,872.39
94430	2001/01/02	\$3,033.38
94410	2001/01/02	\$2,337.28

Example: Applying Conditional Formatting Based on a Hidden (NOPRINT) Field's Values

This example illustrates how to apply conditional formatting based on the values of a hidden (NOPRINT) field. This report uses conditional formatting to draw attention to those employees who have resigned.

Notice that the WHEN attribute's condition evaluates a field (STATUS) that is hidden in the report. Although the field that is evaluated in the condition must be included in the report request, you can prevent it from displaying in the report by using the NOPRINT option, as shown in the following request.

```
TABLE FILE CENTHR
HEADING
"Employee List for Boston"
" "
"For Pay Levels 5+"
" "
"Resigned Employees Shown in <0>Red Bold"
" "
PRINT LNAME FNAME PAYSACLE STATUS NOPRINT
BY ID_NUM
WHERE PLANT EQ 'BOS' AND PAYSACLE GE 5
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=REPORT, GRID=OFF, $
TYPE=DATA, COLUMN=LNAME,
    COLOR=RED, FONT='Arial', STYLE=BOLD, WHEN=STATUS EQ 'RESIGNED', $
TYPE=DATA, COLUMN=FNAME,
    COLOR=RED, FONT='Arial', STYLE=BOLD, WHEN=STATUS EQ 'RESIGNED',
$TYPE=HEADING, FONT='Arial', STYLE=BOLD, SIZE=11, $
TYPE=HEADING, LINE=5, STYLE=-BOLD, $
TYPE=HEADING, LINE=5, ITEM=2, STYLE=BOLD, COLOR=RED, $
ENDSTYLE
END
```

The output is:

Employee List for Boston		
For Pay Levels 5+		
Resigned Employees Shown in Red Bold		
Employee Last <u>ID# Name</u>	First <u>Name</u>	Pay <u>Level</u>
39 GLIOZZO	ANTHONY	5
46 HEBERT	BRYAN	5
70 MIKITKA	MARK	5
79 PALMER	TED	5
85 PHILLIPS	CLAIRE	5
91 ROUSSEAU	DIANE	5
104 WHELEHAN	JAMES	5
129 HAZARD	DAVID	6
142 FLYNN	PAUL	6

Example: Applying Conditional Formatting to a Sort Group

This example illustrates how to apply conditional formatting to a sort group. This report uses conditional formatting to draw attention to those employees who have resigned.

Notice that one conditional declaration is able to apply formatting to all the rows in the sort group. You can accomplish this by evaluating the sort field (STATUS) in the WHEN attribute's condition.

```
TABLE FILE CENTHR
HEADING
"Employee List for Boston"
" "
"For Pay Levels 5+"
" "
PRINT LNAME FNAME PAYSACLE
BY STATUS SKIP-LINE
WHERE PLANT EQ 'BOS' AND PAYSACLE GE 5
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=REPORT, GRID=OFF, $
TYPE=DATA,
    COLOR=RED, FONT='Arial', STYLE=BOLD, WHEN=STATUS EQ 'RESIGNED',$
TYPE=HEADING, FONT='Arial', STYLE=BOLD, SIZE=11, $
ENDSTYLE
END
```

The output is:

Employee List for Boston			
For Pay Levels 5+			
<u>Current Status</u>	<u>Last Name</u>	<u>First Name</u>	<u>Pay Level</u>
DECLINED	ROUSSEAU	DIANE	5
EMPLOYED	MIKITKA	MARK	5
	WHELEHAN	JAMES	5
	FLYNN	PAUL	6
RESIGNED	HEBERT	BRYAN	5
	PHILLIPS	CLAIRE	5
	HAZARD	DAVID	6
TERMINAT	GLIOZZO	ANTHONY	5
	PALMER	TED	5

In order to apply the same conditional formatting to only two columns, instead of all the columns, this version of the report request uses two declarations, each specifying a different column (LNAME and FNAME):

```
TABLE FILE CENTHR
HEADING
"Employee List for Boston"
" "
"Pay Levels 5+"
" "

PRINT LNAME FNAME PAYSACLE
BY STATUS SKIP-LINE
WHERE PLANT EQ 'BOS' AND PAYSACLE GE 5
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=REPORT, GRID=OFF, $
TYPE=DATA, COLUMN=LNAME,
    COLOR=RED, FONT='Arial', STYLE=BOLD, WHEN=STATUS EQ 'RESIGNED', $
TYPE=DATA, COLUMN=FNAME,
    COLOR=RED, FONT='Arial', STYLE=BOLD, WHEN=STATUS EQ 'RESIGNED', $
TYPE=HEADING, FONT='Arial', STYLE=BOLD, SIZE=11, $
ENDSTYLE
END
```

The output is:

Employee List for Boston			
For Pay Levels 5+			
<u>Current Status</u>	<u>Last Name</u>	<u>First Name</u>	<u>Pay Level</u>
DECLINED	ROUSSEAU	DIANE	5
EMPLOYED	MIKITKA	MARK	5
	WHELEHAN	JAMES	5
	FLYNN	PAUL	6
RESIGNED	HEBERT	BRYAN	5
	PHILLIPS	CLAIRE	5
	HAZARD	DAVID	6
TERMINAT	GLIOZZO	ANTHONY	5
	PALMER	TED	5

Example: Using Sequential Conditional Formatting

This example illustrates how to apply sequential conditional formatting to a report. This report uses sequential conditional logic to format each row based on its order total (LINEPRICE).

```
TABLE FILE CENTORD
HEADING
"Order Revenue"
" "
SUM ORDER_DATE LINEPRICE AS 'Order,Total:'
BY HIGHEST 10 ORDER_NUM
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=REPORT, GRID=OFF, $
1. TYPE=DATA, BACKCOLOR=AQUA, STYLE=BOLD+ITALIC,
WHEN=LINEPRICE GT 600000, $
2. TYPE=DATA, BACKCOLOR=YELLOW, STYLE=BOLD,
WHEN=LINEPRICE GT 400000, $
3. TYPE=DATA, BACKCOLOR=ORANGE, STYLE=ITALIC,
WHEN=LINEPRICE GT 200000, $
4. TYPE=DATA, BACKCOLOR=SILVER, FONT='Arial', $
TYPE=HEADING, FONT='Arial', STYLE=BOLD, SIZE=11, $
ENDSTYLE
END
```

Notice that:

1. The first conditional declaration formats any rows whose order total is greater than \$600,000.
2. The second conditional declaration formats any rows whose order total is greater than \$400,000 and less than or equal to \$600,000, as rows with an order total greater than \$200,000 would have already been formatted by the first conditional declaration.
3. The third conditional declaration formats any rows whose order total is greater than \$200,000 and less than or equal to \$400,000, as rows with an order total greater than \$150,000 would have already been formatted by one of the first two conditional declarations.
4. The unconditional declaration following the conditional declarations specifies:
 - Background color, which is also specified by the conditional declarations. It applies background color (silver) to any rows whose order total is less than or equal to \$200,000, since those rows have not already been formatted by the conditional declarations.
 - Font, which is *not* specified by the conditional declarations. It applies font (Arial) to all data rows.

The output is:

Order Revenue

Order Number:	Date Of Order:	Order Total:
94710	2001/01/02	\$406,964.24
94680	2001/01/02	\$421,916.60
94670	2001/01/02	\$513,868.76
94550	2001/01/02	\$496,323.64
94530	2001/01/02	\$3,472.41
94520	2001/01/02	\$261,808.72
94490	2000/12/31	\$633,723.06
94460	2001/01/02	\$3,872.39
94430	2001/01/02	\$3,033.38
94410	2001/01/02	\$2,337.28

Example: Applying Conditional Formatting to Forecasted Values

The following illustrates how you can apply conditional formatting to forecasted values in a report.

```

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 EQ 'Coffee'
ON PERIOD RECAP MOVAVE/D10.1= FORECAST(DOLLARS,1,3,'MOVAVE',3);
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=REPORT, BACKCOLOR=SILVER, WHEN=FORECAST, $
END
    
```

The output is:

<u>Category</u>	<u>PERIOD</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>	<u>MOVAVE</u>
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

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.

In the following StyleSheet declaration, 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=COUNTRY EQ 'ENGLAND', $
```

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.

Example: Field Interaction When Using ACROSSCOLUMN and WHEN

In this request, the RETAIL_COST column under each value of COUNTRY may be printed in italic with an aqua background color, depending on the corresponding value of DIFF:

```
TABLE FILE CAR
SUM RETAIL_COST AND DEALER_COST
COMPUTE DIFF/D12.2=RETAIL_COST - DEALER_COST;
ACROSS COUNTRY
WHERE COUNTRY EQ 'ENGLAND' OR 'FRANCE'
ON TABLE SET SQUEEZE ON
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML AS NF958068
ON TABLE SET STYLE *
TYPE=REPORT, FONT=TIMES, SIZE=10,$
TYPE=REPORT, GRID=OFF,$
TYPE=DATA, ACROSSCOLUMN=RETAIL_COST, BACKCOLOR=AQUA,
WHEN=DIFF GT 5000, STYLE=ITALIC,$
ENDSTYLE
END
```

The output is:

COUNTRY		FRANCE			
ENGLAND	DEALER_COST	DIFF	RETAIL_COST	DEALER_COST	DIFF
45,319	37,853	7,466.00	5,610	4,631	979.00

To specify the DIFF field outside the ACROSS, use the notation C3:

```
WHEN=C3 GT 9000
```

Example: Using ACROSSCOLUMN With WHEN

In this example, SEATS is an ACROSSCOLUMN under COUNTRY. Each value of SEATS is styled differently as a result of the WHEN conditions:

```
TABLE FILE CAR
PRINT SALES SEATS ACROSS COUNTRY
ON TABLE SET SQUEEZE ON
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, ORIENTATION=LANDSCAPE, FONT=COURIER, $
TYPE=DATA, ACROSSCOLUMN=SEATS, WHEN=SEATS EQ 4, STYLE=BOLD,
  COLOR=BLUE, $
TYPE=DATA, ACROSSCOLUMN=SEATS, WHEN=SEATS GT 4, SIZE=12,
  COLOR=AQUA, $
TYPE=DATA, ACROSSCOLUMN=SEATS, WHEN=SEATS LT 4, STYLE=ITALIC,
  COLOR=RED, $
ENDSTYLE
END
```

The output is:

COUNTRY									
ENGLAND		FRANCE		ITALY		JAPAN		W GERMANY	
SALES	SEATS	SALES	SEATS	SALES	SEATS	SALES	SEATS	SALES	SEATS
0	4	-	-	-	-	-	-	-	-
12000	5	-	-	-	-	-	-	-	-
0	4	-	-	-	-	-	-	-	-
0	2	-	-	-	-	-	-	-	-
-	-	-	-	-	-	43000	4	-	-
-	-	-	-	-	-	35030	4	-	-
-	-	-	-	12400	2	-	-	-	-
-	-	-	-	13000	2	-	-	-	-
-	-	-	-	4800	4	-	-	-	-
-	-	-	-	0	2	-	-	-	-
-	-	-	-	-	-	-	-	7800	5
-	-	-	-	-	-	-	-	8950	5
-	-	-	-	-	-	-	-	8900	4
-	-	-	-	-	-	-	-	14000	5
-	-	-	-	-	-	-	-	18940	5
-	-	-	-	-	-	-	-	14000	5
-	-	-	-	-	-	-	-	15600	5
-	-	0	5	-	-	-	-	-	-

Using Conditional Grid Formatting in a Field

You can use conditional grid formatting in order to emphasize a particular cell or field in a PDF or PostScript report.

Example: Creating a Report Using Conditional Grid Formatting

```
TABLE FILE CAR
SUM SALES BY CAR
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
TYPE=DATA, COLUMN=SALES, GRID=HEAVY, WHEN=CAR EQ 'DATSUN', $
ENDSTYLE
END
```

The output is:

<u>CAR</u>	<u>SALES</u>
ALFA ROMEO	30200
AUDI	7800
BMW	80390
DATSUN	43000
JAGUAR	12000
JENSEN	0
MASERATI	0
PEUGEOT	0
TOYOTA	35030
TRIUMPH	0

12 | Cascading Style Sheets

Cascading Style Sheets (CSS) provide a standardized method for styling HTML documents. This W3C-authorized specification requires the use of a Web browser that fully supports CSS.

To use an existing Cascading Style Sheet, simply link it to your report and, optionally, apply additional CSS classes to specific report components using the CLASS attribute. You can create or edit Cascading Style Sheets with a text editor, or use a third-party Web page development tool.

Topics:

- ❑ [What Are Cascading Style Sheets?](#)
- ❑ [Cascading Style Sheets and Precedence Rules](#)
- ❑ [Cascading Style Sheet Formatting Statements: Rules and Classes](#)
- ❑ [Generating an Internal Cascading Style Sheet](#)
- ❑ [Working With External Cascading Style Sheets](#)
- ❑ [Combining CSS Styling With Other Formatting Methods](#)
- ❑ [Linking to an External Cascading Style Sheet](#)
- ❑ [FAQ About Using External Cascading Style Sheets](#)
- ❑ [Troubleshooting Cascading Style Sheets](#)

What Are Cascading Style Sheets?

In this section:

Benefits of Cascading Style Sheets

The Notion of Browser Dependence

Types of Cascading Style Sheets

The World Wide Web Consortium's (W3C) Cascading Style Sheets (CSS) specification defines a simple language for adding styling (such as fonts, colors, and spacing) to HTML documents. A style sheet separates the structural content of a document, defined in HTML, from the styling instructions, which are specified in a CSS. Each style sheet consists of one or more instructions or rules, called statements. Each statement includes a selector that tells a browser which elements on a page are affected by that statement, and a declaration that tells the browser how to draw or render them.

Benefits of Cascading Style Sheets

The benefits of using Cascading Style Sheets to format reports include:

- ❑ **Increased formatting options.** Almost any formatting that you can specify in a Cascading Style Sheet can be applied to a report, including options unavailable using native FOCUS StyleSheet attributes.
- ❑ **Reduced transmission bandwidth.** Cascading Style Sheets enable FOCUS to generate concise HTML output, reducing the bandwidth and overhead required to send output to the browser and display it.
- ❑ **Reduced effort.** Experienced designers have the option of applying CSS to their FOCUS reports as well, avoiding duplications in specifying and maintaining inline formatting instructions.
- ❑ **Easier standards conformance.** Produce consistent documents for sites more easily by specifying default formatting for all Web documents of a similar type in a single Cascading Style Sheet (as opposed to selectively replicating the formatting in inline FOCUS StyleSheets).

The Notion of Browser Dependence

In preparing to use CSS, it is important to understand the pivotal role played by the Web browser. It is the browser's support and implementation of CSS, and not FOCUS, that determines how (or if) a style sheet formats a styled report. Some browsers support CSS specifications fully, while others support only certain versions or formatting attributes, and some offer no support at all. Make sure that your browser and your readers' browsers all support CSS before proceeding.

Types of Cascading Style Sheets

When you create a HOLD file in HTML format, FOCUS generates most of the report output as an HTML table, placing each report item in a separate cell. Through the CSS feature, you can expedite the translation process and minimize the size of the generated HTML file by including an internal (or "embedded") CSS in your request.

You can employ several types of Cascading Style Sheets with FOCUS:

- ❑ **Internal Cascading Style Sheets** that are stored internally in the <head> element of the HTML documents that they format. When you include an internal CSS, FOCUS interprets your FOCUS StyleSheet instructions in a Cascading Style Sheet that is stored in the metadata of your styled HTML output file. The viewing browser interprets this CSS to determine how to render the file contents.

- ❑ **External Cascading Style Sheets** that you or someone else creates are stored in separate files accessible to the target browsers. Users may share documents linked to them. Specify an external CSS location in a FOCUS request using the CSSURL FOCUS StyleSheet attribute or the CSSURL SET parameter.

An external CSS is ideal for defining corporate-standard default styles (such as overall font/size), which users can then selectively override through FOCUS StyleSheet attributes when adjusting styles for particular report components.

- ❑ **Inline Style Sheets** are stored within the tag of an HTML element or within the <head> of the document. These are generally not recommended, as they defeat the basic aim of using style sheets, which is the separation of content and style. They are supported, however, and can be used to override default style values in all other style sheets, as described in the next section.

Cascading Style Sheets and Precedence Rules

As the term "cascading" *implies*, you can apply several style sheets to a single document at the same time. For example, you may associate one style sheet with a document itself, link another style sheet to the first, and then associate a third with the Web browser on which the document is displayed. When multiple style sheets are in effect, they are applied to the document in a predetermined sequence set by the browser (not by FOCUS). The formatting cascades from one style sheet to the next. The precedence of style sheet methods, from highest priority to lowest, is as follows:

- 1.** Inline Style.
- 2.** Internal Style Sheet.
- 3.** External Style Sheet.
- 4.** Browser default.

Inline styles are physically defined within specific HTML elements, and cannot be overridden. An internal Style Sheet, whether coded by you or generated by FOCUS, consists of formatting declarations placed within the <head> tag of the generated HTML file. It is the way you typically override styling defaults established in an external Cascading Style Sheet that resides on a corporate LAN server, and serves as a basic Web document template. External Cascading Style Sheets offer an excellent mechanism for centralizing control of corporate publications throughout a site, but also save you the effort of repeating basic styling instructions from one report request to the next. Finally, you can set your Web browser to observe or ignore Cascading Style Sheets, assuming it is capable of supporting them in the first place. You and the readers of any documents styled with CSS must have CSS-enabled browsers to view or work with them.

The actual process for using Cascading Style Sheets in mainframe FOCUS involves several steps. First, in a FOCUS session specify (or link to) the style attributes that you wish to apply, and create a HOLD file in HTML format. The rest of this chapter discusses formatting methods and alternatives. After running your request, leave FOCUS and transfer the HOLD file to your browser using FTP or another file transfer protocol. Then open the HTML file in your browser and view and work with the HTML file contents.

Cascading Style Sheet Formatting Statements: Rules and Classes

In this section:

- Selecting a CSS Rule
- Naming CSS Classes
- Inheritance and CSS

Cascading Style Sheets (CSS) define formatting in statements called *rules*. For example, this simple rule makes the background color of the body of an HTML page yellow:

```
BODY {background: yellow}
```

Each rule has a selector (BODY in this example) and a declaration (such as background: yellow). A declaration has a property (background) and a value assigned to the property (yellow). A declaration defines formatting, and a selector determines to what it applies. The selector can be any HTML element, or a class. You can define a class by creating a rule for it.

You define classes in a Cascading Style Sheet, and then format report components by assigning CSS classes to them. Define different formatting for the same element by creating several classes for it. For example, if you wish to differentiate between text in sort columns, aggregate columns and detail columns, you can accomplish this by creating three separate classes of the BODY element "sortColumn, aggregateColumn, and detailColumn:

```
BODY.sortColumn {color: blue}
BODY.aggregateColumn {color: green}
BODY.detailColumn {color: black}
```

You can also define generic classes that are not limited to a single element. For example:

```
.pageFooting {font-weight: bolder}
```

You can use generic classes to specify formatting for any FOCUS report component.

Selecting a CSS Rule

When formatting a report, you have the choice of using BODY or TD rules for the entire report, or applying generic class rules to style individual components.

In choosing between the rules for BODY or TD, note that a rule for:

- ❑ **BODY** specifies default formatting for the entire Web page in which the report appears, including for the report itself. Note that this relies on CSS inheritance, which is Web browser-dependent.

If you wish to use Cascading Style Sheets to format a report in the usual way, set STYLEMODE to FULL (the default) or PAGED. If you set it to FIXED and link to an external Cascading Style Sheet, the report inherits formatting from the BODY and PRE elements, but you are unable to format the report using classes and the TD element.

- ❑ **TD** specifies default formatting only for the report, and for any other table cells on the page. (TD stands for table data, the table cell element. FOCUS generates most HTML report output as an HTML table, placing each report item in a separate cell. This enables a TD rule to format the entire report.)

When formatting a report component using a class rule, use a FOCUS StyleSheet to assign the class to the component using the CLASS attribute. When applying several CSS properties to the same report component, it is more efficient to declare them in a single class.

Naming CSS Classes

Class names are case-sensitive. It is recommended to name classes after the functions they perform, not the appearances of the components to which they are applied, so that the names remain meaningful even if the report changes appearance. For example, if you want all report titles to be red, you may name the class declared to format titles Title, but preferably not Red.

Inheritance and CSS

Components in reports formatted using an external Cascading Style Sheet inherit formatting from the TD element and from all elements within which the element nests, such as BODY. (Inheritance, like all CSS behavior, is implemented by the user's Web browser and is browser-dependent.)

This differs from inheritance in a FOCUS StyleSheet, in which report components inherit formatting from higher-level components. When you format reports using external Cascading Style Sheet classes, classes assigned to a report component do not inherit formatting from classes of higher-level components.

Example: Inheriting Report Column Formatting From a TD Element

This report lists vendors that supply products to Gotham Grinds. Its formatting instructions specify that:

- ❑ The default background color for the entire report is orange. This is specified in a rule for the TD element.
- ❑ The data is displayed in an italic Arial font. The data inherits the orange background color from the rule for TD.

- ❑ The PRODUCT_ID data has a yellow background, overriding the default specified in the rule for TD. If the report formatting were specified in a FOCUS StyleSheet instead of in an external CSS, PRODUCT_ID would inherit the italic Arial font from its parent report component, the report data. Instead, because the formatting is specified in an external CSS, PRODUCT_ID inherits its formatting from the rule for the TD element.

The report request and inline FOCUS StyleSheet follow:

```
TABLE FILE GGPRODS
PRINT PRODUCT_DESCRIPTION VENDOR_NAME
BY PRODUCT_ID
ON TABLE SET PAGE-NUM OFF

ON TABLE HOLD AS CSSEXAM1 FORMAT HTML

ON TABLE SET STYLESHEET *
1. TYPE=REPORT, CSSURL = c:\Projects\report02.css, $
2. TYPE=DATA, CLASS=Data, $
3. TYPE=DATA, COLUMN=PRODUCT_ID, CLASS=Sort, $
   ENDSTYLE

END
```

The external Cascading Style Sheet, report02.css, follows:

```
4. TD {background:orange; border:0}
5. TABLE {border:0}
6. .Data {font-style:italic; font-family:Arial}
7. .Sort {background:yellow}
```

1. Set CSSURL to link to the external Cascading Style Sheet report02.css. Alternatively, you can link to a URL on a Web server, as in the following example:

```
TYPE=REPORT, CSSURL=http://webserv1/CSS/reportstyles.css, $
```

2. Specify report data formatting using the CSS rule for the Data class.
3. Specify PRODUCT_ID data formatting using the CSS rule for the Sort class. This overrides the general declaration for formatting report data in line 2.
4. This CSS rule for the TD element specifies an orange background. Because this rule is for the TD, it is applied to the entire report. You can override TD formatting for a particular report component by applying a rule for a generic class to it, as is done in this procedure with the Sort class rule (see line 7).
5. CSS rules for the TD and TABLE elements remove the default grid from the report.
6. This CSS rule for the generic class Data specifies an Arial font family and an italic style. The FOCUS StyleSheet applies this to the report(tm)s data (see line 2). This rule inherits its background color from the rule for the TD element (line 4).

7. This CSS rule for the generic class Sort specifies a yellow background. The FOCUS StyleSheet applies this rule to PRODUCT_ID data (see line 3).

This rule overrides the default background color specified in line 4.

The output is:

Product Code	Product	Vendor Name
E141	Hazelnut	Coffee Connection
E142	French Roast	European Specialities
E144	Kona	Evelina Imports, Ltd
F101	Secoie	Ridgewood Bakeries
F102	Biscuiti	Delarivey Bakeries
F103	Croissant	West Side Bakers
G100	Mug	NY Ceramic Supply
G104	Thermos	ThermoTech, Inc
G110	Coffee Grinder	Appliance Craft
G121	Coffee Pot	Appliance Craft

Generating an Internal Cascading Style Sheet

In this section:

Selecting a Unit of Measurement

When you select HTML format for your report, you instruct FOCUS to generate HTML code to specify its formatting. You can optimize this process by generating an internal Cascading Style Sheet as part of the output, in which case FOCUS places an internal Cascading Style Sheet in the <head> of the HTML document. This, in turn:

- ❑ **Significantly reduces the size of the HTML file generated**, which, in turn, decreases the necessary transmission bandwidth and expedites the display of large reports.
- ❑ **Expands the options for formatting the HTML report.** Certain FOCUS StyleSheet attributes are only supported for HTML reports with internal Cascading Style Sheets. These include the UNITS, BOTTOMMARGIN, TOPMARGIN, LEFTMARGIN, RIGHTMARGIN, SIZE, POSITION, WRAP, and PAGECOLOR attributes. Internal Cascading Style Sheets also permit the addition or removal of underlines from most report components, and allow specification of exact starting positions and sizes of images.

Selecting a Unit of Measurement

How to:

Generate an Internal Cascading Style Sheet

You can choose inches, centimeters, or points as the unit of measurement for page margins and column widths in HTML reports that include an internal Cascading Style Sheet.

To set the unit of measure, use one of the following:

- ❑ **StyleSheet:** UNITS attribute.
- ❑ **SET command:** UNITS parameter.

If you change the unit, existing measurements are automatically converted to the new scale. For example, if you set UNITS to inches and set the top margin to 1, and later change UNITS to centimeters, the top margin automatically converts to 1 centimeter.

Syntax: **How to Generate an Internal Cascading Style Sheet**

To generate an internal Cascading Style Sheet as part of the HTML report output, do the following:

- ❑ Outside a report request, use:

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

- ❑ Within a report request, use:

```
ON TABLE SET HTMLCSS {ON|OFF}
```

where:

ON

Generates an internal Cascading Style Sheet in the HTML output to control most aspects of the report's appearance.

OFF

Turns off the generation of an internal Cascading Style Sheet. Instead, formatting tags are placed in each HTML table cell used to create the report. This is the default.

When generating an internal Cascading Style Sheet, you can also apply an external one in the same request. Should the formatting instructions for a report component conflict, the internal Cascading Style Sheet specifications override the corresponding specifications in the external Cascading Style Sheet, thus providing a mechanism for runtime adjustment of external CSS settings.

Working With External Cascading Style Sheets

In this section:

Applying CSS Styles

Using an External CSS - A Graphical Overview

To format a report using an external Cascading Style Sheet (CSS), you can:

- ❑ **Apply an existing CSS without changes.** Some sites use a single style sheet for all pages on their web site. The same style sheet could also be used for other types of documents.
- ❑ **Edit an existing CSS, adding or modifying rules.** For example, you can add generic classes to an existing style sheet to format your report's components. When you edit an existing CSS, the next time someone displays the report on a browser it reflects those changes without their having to rerun the report (provided an earlier version was not cached on the browser, in which case he or she has to refresh the view to see the edited version).
- ❑ **Create your own Cascading Style Sheet to format reports.**

Applying CSS Styles

How to:

Use the CLASS Attribute to Apply CSS Formatting

You can apply external Cascading Style Sheet (CSS) formatting to:

- ❑ **A report component** (for example, to make a column italic). Assign Cascading Style Sheet classes to report components using FOCUS StyleSheet CLASS attributes. When formatting tabular or free-form reports, you can format any report component by assigning CSS classes. To center headings or footings, use the CENTER option of the HEADING or FOOTING command, rather than doing this in a style sheet.
- ❑ **An entire report** (for example, to make an entire report italic). You can specify this formatting in an external CSS rule for the BODY or TD elements. You do not need a rule for a class of an element, nor do you need a FOCUS StyleSheet declaration when formatting an entire report with this approach.

When using an external Cascading Style Sheet to format a report, it is recommended not to use a FOCUS StyleSheet to specify the report's formatting unless you also generate an internal Cascading Style Sheet.

Syntax: How to Use the CLASS Attribute to Apply CSS Formatting

To apply an external Cascading Style Sheet (CSS) class to a report component, use the following syntax in a FOCUS StyleSheet declaration.

```
TYPE = type, [subtype,] CLASS = classname, [when,] [link,] $
```

where:

type

Identifies the report component to which you are applying the class's formatting. For tabular and free-form reports, it can be any component.

Each report component can be formatted by one class. If you specify several classes for a report component:

- 1.** Classes in declarations with conditional formatting are evaluated first. For each cell in the report component, the first class whose condition is satisfied by the cell(tm)s row is assigned to the cell.
- 2.** If there are no conditional declarations, or if no conditions are satisfied, the class in the first unconditional declaration is assigned to the report component. All subsequent declarations for that component are ignored.

subtype

Is an optional attribute and value needed to specify certain kinds of report components completely. For example, COLUMN and a column identifier are needed to specify a particular report column.

classname

Is the name of the Cascading Style Sheet class you are applying to format the report component. You can assign the same class to multiple report components.

Class names are case sensitive, and must agree with the case name in the class rule in the Cascading Style Sheet. (Note, however, that not all Web browsers enforce case sensitivity for class names.)

when

Is an optional WHEN attribute and value. Supply this to apply conditional formatting.

link

Is an optional URL or JAVASCRIPT attribute and value. Supply this if you want to link the report component to another resource.

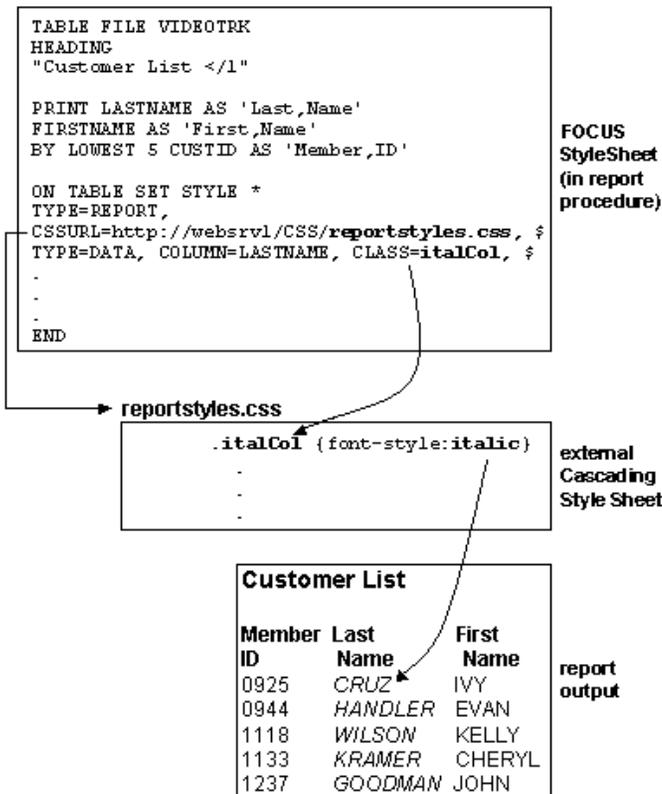
Using an External CSS - A Graphical Overview

How to:
Format a Tabular Report With an External CSS

Three items are required to style a report with an external Cascading Style Sheet (CSS):

- ❑ **An external Cascading Style Sheet** that specifies formatting to be applied.
- ❑ **A FOCUS StyleSheet** in which you apply external CSS formatting attributes to your report components (this is not required if you format the entire report with the CSS).
- ❑ **A link to the external Cascading Style Sheet** from the report.

This example demonstrates the interaction.



Procedure: How to Format a Tabular Report With an External CSS

To format a report using an external Cascading Style Sheet (CSS):

1. Specify the report formatting in the CSS. To specify formatting for:

- ❑ **A report component**, use a rule for any generic class (one not tied to an element). This Cascading Style Sheet rule declares the ColumnTitle generic class:

```
.ColumnTitle {font-family:helvetica; font-weight:bold;
color:blue;}
```

- ❑ **The entire report**, use a rule for the BODY or TD elements (not for a class of these elements), and skip Step 2. This is an effective way to specify default report formatting, and generates more efficient report output than applying a CSS class to the entire report. This Cascading Style Sheet rule for the TD element specifies the element's font family:

```
TD {font-family:helvetica}
```

Because this rule is for the TD element, the formatting is applied to an entire report, not just a report component.

2. Assign classes to report components. In a FOCUS StyleSheet, use the CLASS attribute to assign a Cascading Style Sheet class to each report component that you wish to format. You can assign each component a different class, and you can assign the same class to multiple components. This FOCUS StyleSheet example formats ACROSS values by applying the formatting for the ColumnTitle class:

```
TYPE=AcrossValue, CLASS=ColumnTitle, $
```

3. Link to the CSS. Link the external Cascading Style Sheet by assigning either the URL or the fully qualified pathname for the CSS file, through either the CSSURL FOCUS StyleSheet attribute or the CSSURL SET parameter, as shown below:

- ❑ `TYPE=REPORT, CSSURL = c:\projects\reportstyles.css`
- ❑ `TYPE=REPORT, CSSURL=http://webserv1/css/reportstyles.css`

You can accomplish the same thing using a SET command:

- ❑ `SET CSSURL = c:\projects\reportstyles.css`
- ❑ `SET CSSURL=http://webserv1/css/reportstyles.css`

Example: Formatting a Report Using a Cascading Style Sheet

This annotated report, which displays products currently offered by Gotham Grinds, is formatted using a Cascading Style Sheet (report01.css). The formatting specifies that:

- ❑ The default font family is Arial.
 - The style sheet formatting overrides the report heading default font family of Times New Roman. The heading is also in a larger point size and is center-justified.
- ❑ All column titles are in a bold font and have a light-blue background.
- ❑ When a product's unit price is less than \$27, the report displays that product row in green italics.

The report request and inline FOCUS StyleSheet follow:

```
TABLE FILE GGPRODS
HEADING
"</1 Current Products</1"
PRINT PRODUCT_DESCRIPTION UNIT_PRICE
BY PRODUCT_ID
ON TABLE SET PAGE-NUM OFF

ON TABLE HOLD AS CSSEXAM2 FORMAT HTML
```

1. ON TABLE SET STYLE *
2. TYPE=REPORT, CSSURL=c:\Projects\report01.css, \$
3. TYPE=HEADING, CLASS=headText, \$
4. TYPE=TITLE, CLASS=reportTitles, \$
5. TYPE=DATA, CLASS=lowCost, WHEN=N3 LT 27, \$
6. ENDSTYLE
- END

The external Cascading Style Sheet, report01.css, follows:

```
7. BODY {font-family:Arial, sans-serif}
8. TABLE {border:0}
8. TD {border:0}
9. .reportTitles {font-weight:bolder; background:lightblue;}
10..lowCost {color:green; font-style:italic;}
11..headText {font-family:Times New Roman, serif; font-size:larger;
text-align:center}
```

1. Begin the inline FOCUS StyleSheet.
2. Link to the fully-qualified pathname (or URL, if Web-based) of the external Cascading Style Sheet report01.css.

- 3.** Format the report(tm)s heading using the Cascading Style Sheet(tm)s rule for the headText class.
- 4.** Format the report(tm)s column titles using the CSS(tm)s rule for the reportTitles class.
- 5.** For each report row for which the product(tm)s unit cost is less than \$27, format that row using the CSS rule for the lowCost class.
- 6.** End the inline FOCUS StyleSheet.
- 7.** This CSS rule for the BODY element specifies the font family Arial, and if Arial is unavailable, the generic font family sans serif.

Because this is a rule for BODY, it is applied to the entire report: all text in the report defaults to Arial. You can override this for a particular report component by applying a rule for a generic class to that component, as is done in this procedure with the rule for the headText class (see line 11).

- 8.** These CSS rules for the TABLE and TD elements remove the report(tm)s default grid.
- 9.** This CSS rule for the generic class reportTitles specifies a bolder relative font weight and a light blue background color. The FOCUS StyleSheet applies this to the report(tm)s column titles (see line 4).
- 10.** This CSS rule for the generic class lowCost specifies the text color green and the font style italic. The FOCUS StyleSheet applies this rule conditionally to report rows for which the product(tm)s unit cost is less than \$27 (see line 5).
- 11.** The CSS rule for the generic class headText specifies the font family Times New Roman, and if Times New Roman is unavailable, the generic font family serif. It also specifies a larger relative font size and center justification.

The FOCUS StyleSheet applies this rule to the report(tm)s heading. It overrides the default font family specified in the rule for the BODY element (see line 7).

The output is:

Current Products

Product Code	Product	Unit Price
B141	Hazelnut	58.00
B142	French Roast	81.00
B144	Kona	76.00
<i>F101</i>	<i>Scone</i>	<i>13.00</i>
<i>F102</i>	<i>Biscotti</i>	<i>17.00</i>
F103	Croissant	28.00
<i>G100</i>	<i>Mug</i>	<i>26.00</i>
G104	Thermos	96.00
G110	Coffee Grinder	125.00
G121	Coffee Pot	140.00

Combining CSS Styling With Other Formatting Methods

In this section:

Combining an External CSS With a FOCUS StyleSheet

Combining an External CSS With TABLE Language Formatting

When using a Cascading Style Sheet (CSS) to format a report, you can also apply other formatting methods, such as FOCUS StyleSheets and FOCUS TABLE Language.

Combining an External CSS With a FOCUS StyleSheet

When using an external Cascading Style Sheet (CSS) to format a report, you can always use a FOCUS StyleSheet at the same time, whether or not you generate an internal Cascading Style Sheet.

An effective approach is to link to an external CSS to provide basic report formatting, and then use a FOCUS StyleSheet to selectively override defaults for styling individual report components. Thus, a CSS BODY or TD rule can provide default report formatting, which you can override by providing native FOCUS StyleSheet attributes for those individual report components.

Exceptions: Even when specifying external CSS classes, use native FOCUS StyleSheet attributes to:

- ❑ Create hyperlinks (using URL attributes). However, if you wish to format a hyperlink, do so using the Cascading Style Sheet.
- ❑ Make a FOCUS StyleSheet declaration conditional (using the WHEN attribute).
- ❑ Embed an image (using the IMAGE attribute). If you wish to format the image to position it, do so using the Cascading Style Sheet.

Performance considerations: Unless you also generate an internal Cascading Style Sheet from the FOCUS StyleSheet, combining an external CSS and a FOCUS StyleSheet may reduce the performance benefits associated with the external CSS. This is because you generate more HTML code when styling a report with both external CSS and native FOCUS StyleSheet attributes than if you only use the external CSS. However, this still generates less code than if the report uses only native FOCUS StyleSheet attributes.

Do not double-format. Do not format the same property of the same report component using both an external CSS class (using the CLASS attribute) and a FOCUS StyleSheet attribute, since the class and the StyleSheet attribute could conflict.

For example, do not include the following declarations in the same StyleSheet:

```
TYPE=Data, COLUMN=Country, COLOR=Orange, $
TYPE=Data, CLASS=TextColor, $
```

because both try to assign a color to the report's Country column.

You can specify classes and FOCUS StyleSheet attributes that format different report components, and different properties of the same report component. For example, the following declarations are acceptable in the same StyleSheet:

1. TYPE=Heading, COLOR=Green, \$
1. TYPE=Heading, CLASS=HeadingFontSize, \$
2. TYPE=Data, Column=Country, BACKCOLOR=Yellow, \$
2. TYPE=Data, Column=Car, CLASS=DataBackgroundColor, \$

1. These two declarations are compatible because they format different properties (color and font size).
2. These two declarations are compatible because they format different report components (the Country column and the Car column).

Combining an External CSS With TABLE Language Formatting

TABLE language instructions. You can use TABLE language formatting instructions, such as HEADING CENTER, PAGE-BREAK, and spot markers, but never apply the same formatting to a report component using both a TABLE language instruction and an external Cascading Style Sheet rule. For example, do not specify both of the following for the same report, because both attempt to align the page heading of the report:

- ❑ HEADING CENTER in the report request.
- ❑ Text-align in an external CSS, applied to the report's page heading.

Linking to an External Cascading Style Sheet

How to:

Use the CSSURL SET Parameter to Link to an External CSS

Use the CSSURL Attribute to Link to an External CSS

To format a report using an external Cascading Style Sheet (CSS), you must link to the Cascading Style Sheet by assigning a file location using the CSSURL attribute or parameter:

- ❑ Using CSSURL as a StyleSheet attribute enables you to specify all formatting information in one place, since you can specify a link to the external CSS, and references to CSS classes within the FOCUS StyleSheet. This makes it easier to maintain your formatting logic.
- ❑ Using CSSURL as a SET parameter enables you to switch a link quickly from one CSS to another, which is useful for redirecting many reports at once. To do so, put the SET CSSURL command in its own procedure, and merge it into other report procedures by inserting a -INCLUDE Dialogue Manager command in each.

When the CSSURL is specified in several ways, the most local specification takes precedence. The order of precedence, from local (1) to global (3), is:

1. `TYPE=REPORT, CSSURL = url_or_fully-qualified pathname`
2. `ON TABLE SET CSSURL url_or_fully-qualified pathname`
3. `SET CSSURL = url_or_fully-qualified pathname`

Syntax: How to Use the CSSURL SET Parameter to Link to an External CSS

To link an external Cascading Style Sheet (CSS) to a report using a SET parameter, issue the following command in a procedure:

```
SET CSSURL = css_url_or_fully-qualified pathname
```

or the following ON TABLE SET command in a report request:

```
ON TABLE SET CSSURL css_url_or_fully-qualified pathname
```

where:

css_url_or_fully-qualified pathname

Is the location of the external Cascading Style Sheet (assume it is case-sensitive).

The length limit is:

- ❑ 69 characters in a SET command.
- ❑ 57 characters in an ON TABLE SET command.

Example: Linking to an External Cascading Style Sheet Using the CSSURL Attribute

This report displaying products offered by Gotham Grinds is formatted using an external Cascading Style Sheet (CSS) that is linked through the CSSURL attribute in the FOCUS StyleSheet:

```
TABLE FILE GGPRODS
HEADING
"</1 Current Products</1"
PRINT PRODUCT_DESCRIPTION UNIT_PRICE
BY PRODUCT_ID
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD AS CSSEXAM3 FORMAT HTML

ON TABLE SET STYLESHEET *
TYPE=REPORT, CSSURL = c:\projects\report01.css, $
TYPE=HEADING, CLASS=headText, $
TYPE=TITLE, CLASS=reportTitles, $
TYPE=DATA, CLASS=lowCost, WHEN=N3 LT 27, $
ENDSTYLE
END
```

The output is:

Current Products

Product Code	Product	Unit Price
B141	Hazelnut	58.00
B142	French Roast	81.00
B144	Kona	76.00
<i>F101</i>	<i>Scone</i>	<i>13.00</i>
<i>F102</i>	<i>Biscotti</i>	<i>17.00</i>
F103	Croissant	28.00
<i>G100</i>	<i>Mug</i>	<i>26.00</i>
G104	Thermos	96.00
G110	Coffee Grinder	125.00
G121	Coffee Pot	140.00

Syntax: **How to Use the CSSURL Attribute to Link to an External CSS**

Use the following syntax to link an external Cascading Style Sheet (CSS) to your report using a FOCUS StyleSheet attribute:

```
[TYPE=REPORT,] CSSURL=css_path, $
```

where:

TYPE=REPORT

Specifies that this attribute is being applied to the entire report. If it is omitted, the StyleSheet declaration still defaults to this setting.

css_path

Specifies the URL or fully-qualified path name of the location where the external Cascading Style Sheet definition file resides.

If you specify CSSURL multiple times, the last value specified in an ON TABLE SET command overrides all the other values within that report request. If CSSURL is not specified within a report request, the last value specified with SET overrides all the others.

FAQ About Using External Cascading Style Sheets

This topic answers the most frequently asked questions (FAQ) about using external Cascading Style Sheets (CSS) to format reports.

Does it answer your question? Please send in any questions that you may still have. Each question will get a response, and will also be considered for inclusion in a future release of FAQ. Any comments on this document are also welcome.

You can:

❑ **E-mail them** to books_info@ibi.com. Please include your name and phone number, and put *Cascading Style Sheet FAQ* in the subject line.

❑ **Send them** to:

Documentation Services
Attn: Core Reporting Group
Information Builders
Two Penn Plaza
New York, NY 10121-2898

Please include your name, phone number, e-mail address, and postal address.

How do I specify a report's default formatting using CSS?

Specify default formatting for an entire report in an external Cascading Style Sheet rule for the BODY or TD element.

Do I always need to use the CLASS attribute?

No. You need a CLASS attribute in a FOCUS StyleSheet if you specify formatting for an individual report component. You use CLASS to assign a rule for a generic class to the report component. When you specify formatting for the entire report, you do so in a rule for the BODY or TD element, not in a rule for a class, so you omit the CLASS attribute.

Can I use a Cascading Style Sheet and a FOCUS StyleSheet together?

When linking to an external Cascading Style Sheet, you can also specify native FOCUS StyleSheet attributes in a FOCUS StyleSheet. However, if you do not generate an internal Cascading Style Sheet, you should not specify CSS classes (CLASS=) and native FOCUS StyleSheet attributes in the same FOCUS StyleSheet (except to specify conditional formatting, to specify a link to another resource, or to embed an image).

Which version of CSS does FOCUS support?

Support for different versions of Cascading Style Sheets (such as CSS2) is determined entirely by your Web browser's support and implementation of Cascading Style Sheets, not by FOCUS.

Which types of reports can I format using an external Cascading Style Sheet?

You can format tabular reports, including regular (column-oriented) reports and Financial Modeling Language (FML, also known as extended matrix or row-oriented) reports.

Troubleshooting Cascading Style Sheets

This topic addresses some common problems encountered when formatting reports with Cascading Style Sheets (CSS).

What problems did you encounter? If you have troubleshooting suggestions that you think others will find helpful, you may send them to be considered for inclusion in a future release. You can:

❑ **E-mail them** to books_info@ibi.com. Please include your name and phone number, and put *Cascading Style Sheet troubleshooting for FOCUS applications* in the subject line.

❑ **Send them** to:

Documentation Services
Attn: Core Reporting Group
Information Builders
Two Penn Plaza
New York, NY 10121-2898

Please include your name, phone number, e-mail address, and postal address.

Symptom: The report is not using any of the Cascading Style Sheet's formatting.

❑ **Reason 1:** Your Web browser may not support Cascading Style Sheets.

Solution 1: Check to be sure that your browser supports Cascading Style Sheets; if it does not, install an appropriate browser version.

❑ **Reason 2:** Your Web browser may be set to ignore Cascading Style Sheets.

Solution 2: Reset the browser to accept a document's Cascading Style Sheet. For instructions about checking or changing a browser's setting, see your browser's help.

Symptom: The report reflects some, but not all, of the CSS formatting.

❑ **Reason 1:** Your Web browser's support and implementation of Cascading Style Sheets determines how a Cascading Style Sheet rule formats your report. It has nothing to do with FOCUS. You may experience this symptom because your browser does not support the level of Cascading Style Sheets that you are using, leaving some CSS features unimplemented.

Solution 1: Upgrade your browser to a version that supports all the CSS features used to format the report, or edit the Cascading Style Sheet to remove features that are unsupported by some of the browsers that are used to display the report.

- ❑ **Reason 2:** Your Web browser may be set to use your personal Cascading Style Sheet, and you may have rules there that override those specified in the Cascading Style Sheet assigned to the report. See your third-party CSS documentation.

Solution 2: Reset your browser to accept each document's Cascading Style Sheet, or edit the rules in the two Cascading Style Sheets so that they no longer conflict.

- ❑ **Reason 3:** Certain Web browsers ignore an entire rule if they do not support a property specified in it. If your Web browser does not support a property specified in a rule for one of the classes assigned to the report, none of the report components to which that rule was assigned are formatted.

Solution 3: Remove the unsupported property, or upgrade your browser to a version that supports the property.

- ❑ **Reason 4:** Each report component can be assigned only one Cascading Style Sheet class. If you have specified more than one class, only the first one specified is assigned to the component; the others are ignored.

If a class has not yet been assigned to a report cell, and you specify conditional formatting for it, only the first class whose condition is satisfied by that row is assigned to the cell. The others are ignored.

Solution 4: Do not assign more than one CSS class to each report component. If you need to apply multiple attributes, bundle them into a single class.

- ❑ **Reason 5:** Some Web browsers implement CSS inheritance rules for nested elements in ways that do not conform to the CSS standard. If you are using such a browser, and "for example" you specify some formatting in a rule for the BODY element, your browser may not apply the rule to other elements nested within BODY.

Solution 5: Specify the report's formatting in a rule for a different element (for example, if the browser does not correctly implement inheritance from BODY, use a rule for TD), or else upgrade your browser to a version that correctly supports inheritance.

- ❑ **Reason 6:** External Cascading Style Sheets can be subject to certain restrictions when used with other formatting methods. For example, if a report's FOCUS StyleSheet does not generate an internal Cascading Style Sheet, but it references external CSS classes and also specifies native FOCUS StyleSheet attributes, there may be a formatting conflict.

Solution 6: The solution depends on the kind of formatting conflict. In the example above, the solution is to generate an internal Cascading Style Sheet.

13 | Working With Styled Output Formats

Some advanced features of styled report output depend on whether the report is produced as HTML, PDF, PostScript, Excel 2000, or Excel 97 format. These features are described in this chapter.

For information about creating a StyleSheet, identifying and styling report components, and choosing a styled output format, see Styling Reports.

For advanced StyleSheet techniques and features that apply to styled output formats, see Advanced StyleSheet Features.

Topics:

- ❑ Working With HTML Reports
- ❑ Working With Excel 2000 and Excel 97 Reports
- ❑ Working With PostScript and PDF Reports

Working With HTML Reports

In this section:

Preserving Leading and Internal Blanks in Report Output

Creating HTML Reports With Absolute Positioning

Preserving Leading and Internal Blanks in Report Output

How to:

Preserve Leading and Internal Blanks in HTML and EXL2K Reports

By default, HTML browsers and Excel remove leading and trailing blanks from text and compress multiple internal blanks to a single blank.

If you want to preserve leading and internal blanks in HTML and EXL2K report output, you can issue the SET SHOWBLANKS=ON command.

Even if you issue this command, trailing blanks will not be preserved except in heading, subheading, footing, and subfooting lines that use the default heading or footing alignment.

Syntax: **How to Preserve Leading and Internal Blanks in HTML and EXL2K Reports**

In a FOCEXEC, on the command line, or in a profile use the following syntax

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

In a request, use the following syntax

```
ON TABLE SET SHOWBLANKS {OFF|ON}
```

where:

OFF

Removes leading blanks and compresses internal blanks in HTML and EXL2K report output.

ON

Preserves leading and internal blanks in HTML and EXL2K report output. Also preserves trailing blanks in heading, subheading, footing, and subfooting lines that use the default heading or footing alignment.

Example: Preserving Leading and Internal Blanks in HTML and EXL2K Report Output

The following request creates a virtual field that adds leading blanks to the value ACTION and both leading and internal blanks to the values TRAIN/EX and SCI/FI in the CATEGORY field. It also adds trailing blanks to the value COMEDY:

```
SET SHOWBLANKS = OFF
DEFINE FILE MOVIES
NEWCAT/A30 = IF CATEGORY EQ 'ACTION' THEN ' ACTION'
            ELSE IF CATEGORY EQ 'SCI/FI' THEN 'SCIENCE FICTION'
            ELSE IF CATEGORY EQ 'TRAIN/EX' THEN ' TRAINING EXERCISE'
            ELSE IF CATEGORY EQ 'COMEDY' THEN 'COMEDY '
            ELSE 'GENERAL';
END
TABLE FILE MOVIES
SUM CATEGORY LISTPR/D12.2 COPIES
BY NEWCAT
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
GRID=OFF,$
TYPE=REPORT, FONT=COURIER NEW,$
ENDSTYLE
END
```

With SHOWBLANKS OFF, these additional blanks are removed:

<u>NEWCAT</u>	<u>CATEGORY</u>	<u>LISTPR</u>	<u>COPIES</u>
TRAINING EXERCISE	TRAIN/EX	119.87	10
ACTION	ACTION	94.82	14
COMEDY	COMEDY	154.80	19
GENERAL	MYSTERY	1,216.82	67
SCIENCE FICTION	SCI/FI	114.84	7

With SHOWBLANKS ON, the additional leading and internal blanks are preserved. Note that trailing blanks are not preserved:

<u>NEWCAT</u>	<u>CATEGORY</u>	<u>LISTPR</u>	<u>COPIES</u>
TRAINING	EXERCISE TRAIN/EX	119.87	10
ACTION	ACTION	94.82	14
COMEDY	COMEDY	154.80	19
GENERAL	MYSTERY	1,216.82	67
SCIENCE	FICTION	114.84	7

Creating HTML Reports With Absolute Positioning

How to:

Create Web Archive Report Output

Create a DHTML Report

Reference:

Usage Notes for Format DHTML

Format DHTML provides HTML output that has most of the features normally associated with output formatted for printing such as PDF or PostScript output. You can create an HTML file (.htm) or a Web Archive file (.mht). The type of output file produced is controlled by the value of the HTMLARCHIVE parameter.

Some of the features supported by format DHTML are:

- ❑ **Absolute positioning.** DHTML precisely places text and images inside an HTML report, allowing you to use the same StyleSheet syntax to lay out HTML as you use for PDF or PS output.
- ❑ **On demand paging.** On demand paging is available with SET HTMLARCHIVE=OFF.
- ❑ **PDF StyleSheet features.** For example, the following features are supported: grids, background colors, OVER.

Syntax: **How to Create Web Archive Report Output**

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

where:

ON

Creates output in Web Archive format. The file type of the output file is MHT.

OFF

Creates output in HTML format. The file type of the output file is HTM. OFF is the default value.

Syntax: How to Create a DHTML Report

```
[ON TABLE] HOLD [AS name] FORMAT DHTML
```

where:

name

Specifies the name of the output file. The extension will be HTML if SET HTMLARCHIVE is OFF or MHT if SET HTMLARCHIVE is ON.

Reference: Usage Notes for Format DHTML

- ❑ The font map file for DHTML reports is DHTML FOCFTMAP on z/VM and is member DHTML in the ERRORS PDS on z/OS.
- ❑ Legacy compound reports are not supported.

Example: Creating a DHTML Report

The following example creates a DHTML file that has an image with absolute positioning:

```
SET HTMLARCHIVE = OFF
TABLE FILE GGSALES
SUM UNITS BY CATEGORY BY PRODUCT
ON TABLE SUBHEAD
"Report on Units Sold"
" "
" "
" "
" "
" "
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT DHTML
ON TABLE SET STYLE *
TYPE=TABHEADING,IMAGE=c:\images\GOTHAM.GIF, POSITION=(.25 .25),
SIZE=(.5 .5), $
ENDSTYLE
END
```

The output shows that the look and positioning are the same as they would be for a PDF report:

Report on Units Sold



<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>
Coffee	Capuccino	189217
	Espresso	308986
	Latte	878063
Food	Biscotti	421377
	Croissant	630054
	Scone	333414
Gifts	Coffee Grinder	186534
	Coffee Pot	190695
	Mug	360570
	Thermos	190081

Working With Excel 2000 and Excel 97 Reports

In this section:

- Creating Styled Excel 2000 Files
- National Language Support With EXL2K
- Displaying Formatted Dates and Numeric Values
- Controlling Column Width and Wrapping
- Locking Columns in Excel Report Output
- Using the Excel 2000 Formula Option
- Using the Excel 2000 PIVOT Option
- Designating CACHEFIELDS in PivotTables
- Designating PAGEFIELDS in PivotTables
- Excel Named Ranges
- Identifying Null Values in Excel 2000
- Excel Table of Contents
- Excel Compound Reports
- Transferring Excel 2000 Formatted Files Using FTP
- Creating Styled Excel 97 Files

EXL2K format generates styled reports in Excel 2000 HTML format for use on other platforms and on the Web. Feature options enable FOCUS users to also download all fields mentioned in their requests in an Excel PivotTable, or include interactive Excel formulas for FOCUS aggregation operations for performing additional "what if" analyses on their data within Excel 2000.

EXL97 is an HTML-based HOLD format for generating formatted Excel 97 spreadsheets. EXL97 is a full StyleSheet driver for accurately rendering all report elements (such as headings and subtotals, for example) as well as applying StyleSheet syntax (such as, conditional styling). You must have Microsoft Excel 97 or higher installed on your computer to display an Excel 97 report.

Creating Styled Excel 2000 Files

How to:

Create a Styled Excel 2000 File

EXL2K format is a full StyleSheet driver for accurately rendering all report elements (such as headings and subtotals) as well as applying StyleSheet syntax (such as, conditional styling). The EXL2K format accurately displays formatted dates and numeric values and controls column width and wrapping in Excel 2000.

The three HOLD format options for Excel 2000 are:

- ❑ **HOLD FORMAT EXL2K.** EXL2K format supports full styling of FOCUS report elements (such as use of centered headings, multiple fonts and background colors, and subtotals), permitting you to exploit StyleSheets features such as conditional styling in the generated Excel 2000 spreadsheets.
- ❑ **FORMAT EXL2K FORMULA.** The FORMULA option automatically generates properly formatted Excel formulas for FOCUS summary operations, such as row and column totals and COMPUTE commands.
- ❑ **HOLD FORMAT EXL2K PIVOT.** The PIVOT option generates Excel 2000 spreadsheets in PivotTables, which are Excel tools used to analyze complex data in an OLAP-like environment. This allows users to create differing views of their data by dragging and dropping data fields within the "PivotTable" to employ varied sorts across rows or columns.

Syntax: How to Create a Styled Excel 2000 File

```
[ON TABLE] HOLD [AS filename] FORMAT EXL2K [PIVOT] [FORMULA]
```

where:

EXL2K

Creates an Excel-formatted output file that may include styling based on internal or external StyleSheets features. The file type on VM is XHT; the extension on Windows platforms is .xht;

PIVOT

Creates an output file in Excel PivotTable format with an accompanying PivotTable cache file. The filetype of the Pivot Table file is XML; the extension on Windows platforms is .xml. For more information about this option, see [Using the Excel 2000 PIVOT Option](#) on page 657.

FORMULA

Creates an XHT output file including appropriate Excel formulas for all FOCUS numeric summary operations. For more information about this option, see [Using the Excel 2000 Formula Option](#) on page 651.

Example: Creating an EXL2K Output File

This example shows how to create a styled report in EXL2K format, with conditional styling based on the contents of CENTORD:

```
TABLE FILE CENTORD
HEADING
"LINE COST BY STATE"
SUM LINE_COGS AS 'Cost'
  BY STATE AS 'State'
BY PLANTLNG AS 'Plant'
BY STORENAME AS 'Store Name'
  WHERE TOTAL LINE_COGS GT 10000000
  ON TABLE HOLD AS EXL2K1 FORMAT EXL2K
ON TABLE SET STYLE *
TYPE=REPORT, FONT=ARIAL, TITLETEXT=SALES REPORT, $
TYPE=DATA, COLUMN=LINE_COGS, COLOR=RED, BACKCOLOR=YELLOW,
  WHEN=LINE_COGS GT 20000000,$
TYPE=DATA, COLUMN=STORENAME, BACKCOLOR=YELLOW,
WHEN=LINE_COGS GT 20000000,$
TYPE=HEADING, FONT=ARIAL BLACK, COLOR=RED, BACKCOLOR=SILVER, SIZE=16, $
TYPE=TITLE, FONT=ARIAL, SIZE=12, $
ENDSTYLE
END
```

The output is:

A1 = LINE COST BY STATE				
	A	B	C	D
1	LINE COST BY STATE			
2	State	Plant	Store Name	Cost
3	CA	Los Angeles	eMart	21,820,010.00
4			Audio Expert	11,164,354.00
5	DC	Boston	Audio Expert	53,959,787.00
6	FL	Orlando	TV City	18,243,573.00
7	GA	Orlando	eMart	15,207,484.00
8			Audio Expert	11,990,222.00
9	IL	St Louis	eMart	11,975,226.00
10	LA	Dallas	Audio Expert	11,566,844.00
11	MA	Boston	Audio Expert	22,069,810.00
12	MD	Boston	Audio Expert	19,133,396.00
13	MI	St Louis	eMart	41,581,383.00
14	MN	St Louis	eMart	16,196,564.00
15	MO	St Louis	Audio Expert	11,135,600.00
16	NJ	Boston	AV VideoTown	17,688,958.00
17	NY	Boston	Audio Expert	12,871,729.00
18	OH	St Louis	eMart	15,115,073.00
19	OK	Dallas	eMart	16,491,825.00
20	ON	Seattle	TV City	11,273,596.00
21	PA	Boston	eMart	10,243,372.00
22	TX	Dallas	eMart	19,471,542.00
23			Audio Expert	20,651,750.00

You can also adjust column contents in your Excel spreadsheets using the StyleSheets keywords WRAP (for wrapping column contents) and SQUEEZE (for truncating columns).

National Language Support With EXL2K

How to:

Set the Default Language

Excel 2000 users can select one of six languages as their default language when generating EXL2K formatted output. In addition to English, which is the automatic default, users can issue a SET command to select one of five other options.

Syntax: **How to Set the Default Language**

Excel 2000 users can select one of six languages as their default language when generating EXL2K formatted output. In addition to English, which is the automatic default, you can select one of five other options

```
SET EXL2KLANG=lang
```

where:

lang

Is one of the following: AME, FRE, SPA, GER, JPN or KOR.

You can code the SET EXL2KLANG in your user profile or include it in a FOCEXEC to override the default setting in the NLSCFG ERRORS file for a specific request.

Displaying Formatted Dates and Numeric Values**Reference:**

Usage Notes for Date and Numeric Formats

Using Date Separators in Excel

When translating numeric and date formats from FOCUS to Excel, there must be a corresponding Excel format to translate to. If there is no corresponding format, then the value will be formatted in the closest matching Excel format or in Excel's General format.

Excel 2000 spreadsheets generated by FOCUS contain the numeric formatting specified in the data source Master File or as specified in a temporary field. All FOCUS numeric values and date formats (such as currency and Smart Dates) are translated into supported Excel formats and display properly in Excel 2000.

Example: Displaying Formatted Numeric Data in Excel 2000

This example illustrates how formatted numeric data appears in a spreadsheet when you use the EXL2K format. Note that the format for the LINEPRICE field D12.2M (that represents floating point double-precision with two decimal places, commas, and a floating dollar sign) is translated into the corresponding Excel format.

```
SET PAGE-NUM=OFF
TABLE FILE CENTORD
"Line Total Report"
"Excel 2000 Spreadsheet"
" "
SUM LINEPRICE
BY STATE AS 'State'
BY PLANTLNG AS 'Plant'
BY STORENAME AS 'Store Name'
WHERE TOTAL LINEPRICE FROM 9000000 TO 20000000
ON TABLE SET BYDISPLAY ON
   ON TABLE HOLD AS EXL2K2 FORMAT EXL2K
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, FONT=TAHOMA, $
TYPE=HEADING, SIZE=14, COLOR=NAVY, $
TYPE=HEADING, LINE=2, SIZE=12, COLOR=RED, $
TYPE=TITLE, JUSTIFY=CENTER, STYLE=BOLD, $
TYPE=DATA, JUSTIFY=CENTER, $
TYPE=DATA, COLUMN=LINEPRICE, JUSTIFY=RIGHT, $
END
```

The output is:

A1 = Line Total Report				
	A	B	C	D
1	Line Total Report			
2	Excel 2000 Spreadsheet			
3				
4	State	Plant	Store Name	Line Total
5	CA	Los Angeles	Audio Expert	\$12,579,095.49
6	CT	Boston	TV City	\$11,102,100.95
7	DC	Boston	Consumer Merchandise	\$9,694,317.15
8	FL	Orlando	eMart	\$10,719,000.96
9	GA	Orlando	eMart	\$17,117,862.54
10	GA	Orlando	Audio Expert	\$13,491,356.00
11	IL	St Louis	eMart	\$13,591,117.80
12	IL	St Louis	TV City	\$9,118,166.75
13	IN	St Louis	eMart	\$10,399,132.19
14	LA	Dallas	Audio Expert	\$12,965,359.19
15	MA	Boston	eMart	\$9,219,965.24
16	MN	St Louis	eMart	\$18,614,598.44
17	MN	St Louis	AV VideoTown	\$9,548,014.55
18	MO	St Louis	Audio Expert	\$12,918,789.82
19	NY	Boston	eMart	\$10,508,112.48
20	NY	Boston	Audio Expert	\$14,899,270.96
21	OH	St Louis	eMart	\$17,032,414.42
22	OK	Dallas	eMart	\$18,565,287.49
23	ON	Seattle	TV City	\$12,699,111.42
24	PA	Boston	eMart	\$11,636,918.64

Note the repetition of the sort field values in the output. This presentation is particularly desirable in a spreadsheet and is controlled by the command `ON TABLE SET BYDISPLAY ON`.

Example: Displaying Formatted Dates in Excel 2000

This example illustrates how customized dates display in a spreadsheet when using the EXL2K format.

Month Hired is defined in the request as MtYY format (the month is represented as a three-character abbreviation with an initial capital letter followed by a four-digit year).

Years of Service is defined as I4C format, a four-digit integer with a comma if required. Both formats are properly displayed as defined in the spreadsheet.

```
SET PAGE-NUM=OFF
DEFINE FILE EMPLOYEE
YRHIREDD/YY = HIRE_DATE;
MHIREDD/MtYY = HIRE_DATE;
TOTSVC/I4C = 2002 - YRHIREDD;
END
TABLE FILE EMPLOYEE
"Employee Service Report for 2002"
"Excel 2000 Spreadsheet"
" "
PRINT FIRST_NAME AS 'First Name'
MHIREDD AS 'Month Hired'
TOTSVC AS 'Years of Service'
BY LAST_NAME AS 'Last Name'
ON TABLE SET BYDISPLAY ON
ON TABLE HOLD AS EXL2K3 FORMAT EXL2K
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, FONT=TAHOMA, $
TYPE=HEADING, SIZE=14, COLOR=NAVY, $
TYPE=HEADING, LINE=2, SIZE=12, COLOR=RED, $
TYPE=TITLE, JUSTIFY=CENTER, STYLE=BOLD, $
TYPE=DATA, JUSTIFY=CENTER, $
TYPE=DATA, COLUMN=TOTSVC, COLOR=BLUE, WHEN=TOTSVC GT 20, $
END
```

The output is:

A1 = Employee Service Report for 2002				
	A	B	C	D
1	Employee Service Report for 2002			
2	Excel 2000 Spreadsheet			
3				
4	Last Name	First Name	Month Hired	Years of Service
5	BANNING	JOHN	Aug, 1982	20
6	BLACKWOOD	ROSEMARIE	Apr, 1982	20
7	CROSS	BARBARA	Nov, 1981	21
8	GREENSPAN	MARY	Apr, 1982	20
9	IRVING	JOAN	Jan, 1982	20
10	JONES	DIANE	May, 1982	20
11	MCCOY	JOHN	Jul, 1981	21
12	MCKNIGHT	ROGER	Feb, 1982	20
13	ROMANS	ANTHONY	Jul, 1982	20
14	SMITH	MARY	Jul, 1981	21
15	SMITH	RICHARD	Jan, 1982	20
16	STEVENS	ALFRED	Jun, 1980	22

The command ON TABLE SET BYDISPLAY ON ensures that sort fields are repeated in each spreadsheet cell.

Reference: Usage Notes for Date and Numeric Formats

Warning: The following formats are not supported in EXL2K and may produce unpredictable results when translated into Excel's General Format:

- YY, Y, M, D, JUL, and I2MT.
- Any date format with a Q (quarter).
- Any packed-decimal (P) date formats.
- Any alphanumeric (A) date formats.
- Fixed Dollar (N) formats.
- Multiple format options. Only single format options are supported when using FORMAT EXL2K. For example, the formats I9C and I9B are supported, but I9BC is not.

The following applies to headings and footings with embedded numeric fields:

- ❑ If you embed a numeric field in a heading, subheading, footing, or subfooting of a report in Excel 2000 or higher format, the numeric field displays in Excel general format (text). To display a numeric field in Excel number format, you must set HEADALIGN=BODY in the StyleSheet.

Reference: Using Date Separators in Excel

In order to use a "-" as a separator between month, day, and year in Excel, you must change the default date separator for Windows®. This setting can be located under Regional Options in the Control Panel.

Controlling Column Width and Wrapping

How to:

Wrap Data in Excel 2000

Set Column Width in Excel 2000

You control data wrapping and column widths in FORMAT EXL2K, by:

- ❑ **Turning data wrapping on.** The default behavior is to have all data wrap according to a default column width determined by Excel.
- ❑ **Turning data wrapping off.** This setting allows columns to expand to the length of the data value. The column width is determined by Excel, but should be wide enough to accommodate the longest data value in the column. If a portion of the data is hidden, you can adjust the column width in Excel after the spreadsheet has been generated.
- ❑ **Turning data wrapping off and setting the column width at the same time.** If a data value is wider than the specified column width, a portion of the data will be hidden from view. You can adjust the column width in Excel after the spreadsheet has been generated.
- ❑ **Specifying the exact width of a column with data wrapping on.**

Syntax: How to Wrap Data in Excel 2000

```
TYPE=REPORT, [COLUMN=column,] WRAP=value, $
```

where:

column

Identifies a particular column. If COLUMN is not included in the declaration, the column width specified with SQUEEZE is applied to the entire report.

value

Is one of the following:

ON

Turns on data wrapping. ON is the default. With this setting, the column width is determined by the client (Excel). Data wraps if it exceeds the width of the column and the row's height expands to meet the new height of the wrapped data.

OFF

Turns off data wrapping. This setting adjusts the column width of the largest data value in the column. Data will not wrap in any cell in the column.

n

Represents a specific numeric value for the column width. The value represents the measure specified with the UNITS parameter (the default is inches). This is the most commonly used SQUEEZE setting in an Excel 2000 report.

Syntax: How to Set Column Width in Excel 2000

```
TYPE=REPORT, [COLUMN=column,] SQUEEZE=n, $
```

where:

column

Identifies a particular column. If COLUMN is not included in the declaration, the column width specified with SQUEEZE is applied to the entire report.

n

Represents a specific numeric value for the column width. The value represents the measure specified with the UNITS parameter (the default is inches). This is the most commonly used SQUEEZE setting in an Excel 2000 report.

Note: SQUEEZE=(ON/OFF), which turns data wrapping on and off, is not supported for EXL2K, so if a data value is wider than the specified column width, it will be hidden from view. However, you can adjust column widths in Excel after you generate a spreadsheet.

Example: Controlling Column Width and Wrapping in Excel 2000

The following example illustrates how to turn on and turn off data wrapping in a column and how to set the column width for a particular column. The UNITS in this example are set to inches (the default).

```
DEFINE FILE CENTORD
MYDATE/MDY='10/22/60';
RCD/D14.3=LINE_COGS;
VERYLONG/A80='Multiply quantity times line_cost to'|
' calculate line_cost_of_goods';
END
TABLE FILE CENTORD
SUM MYDATE RCD
VERYLONG AS 'Default' VERYLONG AS 'WRAP=OFF'
VERYLONG AS 'WRAP=4.1' VERYLONG AS 'WRAP=2'
VERYLONG AS 'SQUEEZE=2' LINE_COGS
BY REGION AS 'Region'
ON TABLE HOLD AS EXL2K4 FORMAT EXL2K
ON TABLE SET STYLE *
TYPE=DATA, COLUMN=MYDATE, JUSTIFY=CENTER, $
1. TYPE=REPORT, COLUMN=VERYLONG(2), WRAP=OFF, $
2. TYPE=REPORT, COLUMN=VERYLONG(3), WRAP=4.1, $
3. TYPE=REPORT, COLUMN=VERYLONG(4), WRAP=2, $
4. TYPE=REPORT, COLUMN=VERYLONG(5), SQUEEZE=2, $
END
```

where:

1. Identifies the column titled "WRAP=OFF" and turns off data wrapping for that column.
2. Identifies the column titled "WRAP=4.1" and sets the column width to 4.1 inches with data wrapping on.
3. Identifies the column titled "WRAP=2" and sets the column width to 2 inches with data wrapping on.
4. Identifies the column titled "SQUEEZE=2" and sets the column width to 2 inches with data wrapping off.

Note: The column titled "Default" illustrates the default column width and wrapping behavior.

Since the output is wider than this page, it is shown in two sections. The following output displays the "Default", "WRAP=OFF", and "WRAP=4.1" columns:

REGION	MYDATE	RCD	DEFAULT	WRAP=OFF	WRAP=4.1
EAST	10/22/60	233,492,476.000	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS
NORTH	10/22/60	156,931,554.000	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS
SOUTH	10/22/60	184,491,442.000	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS
WEST	10/22/60	71,649,432.000	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS

The following output displays the "WRAP=2", and "SQUEEZE=2" columns:

WRAP=2	SQUEEZE=2	Line Cost Of Goods Sold
MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LIF	233,492,476.00
MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LIF	156,931,554.00
MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LIF	184,491,442.00
MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LIF	71,649,432.00

Locking Columns in Excel Report Output

How to:

- Enable Spreadsheet Locking
- Lock Specific Cells Within a Spreadsheet

Using StyleSheet attributes, you can lock Excel spreadsheet values so they are read-only. These attributes apply to all Excel formats including EXL2K, EXL2K PIVOT, and EXL2K FORMULA.

Syntax: **How to Enable Spreadsheet Locking**

To enable locking, use the following attributes:

```
TYPE=REPORT, PROTECTED={ON|OFF}, [ LOCKED={ON|OFF} ], $
```

where:

```
TYPE=REPORT, PROTECTED=ON
```

Is necessary to enable spreadsheet locking. PROTECTED=OFF is the default. If you omit the LOCKED=OFF attribute, the entire spreadsheet is locked.

```
LOCKED=ON
```

Locks the entire spreadsheet. ON is the default value.

```
LOCKED=OFF
```

Unlocks the spreadsheet as a whole, but enables you to lock or unlock specific cells or groups of cells.

Syntax: **How to Lock Specific Cells Within a Spreadsheet**

Once you include the following declaration in your StyleSheet, you can specify the LOCKED attribute for specific cells or groups of cells:

```
TYPE=REPORT, PROTECTED=ON, LOCKED=OFF, $
```

To lock specific parts of the spreadsheet, add the LOCKED=ON attribute to the StyleSheet declaration for the cells you want to lock.

```
TYPE=type, [ COLUMN=columnspec ], LOCKED={ON|OFF}, $
```

where:

type

Is the type of element that describes the cells to be locked.

columnspec

Is a valid column specification.

Example: **Locking an Entire Excel Spreadsheet**

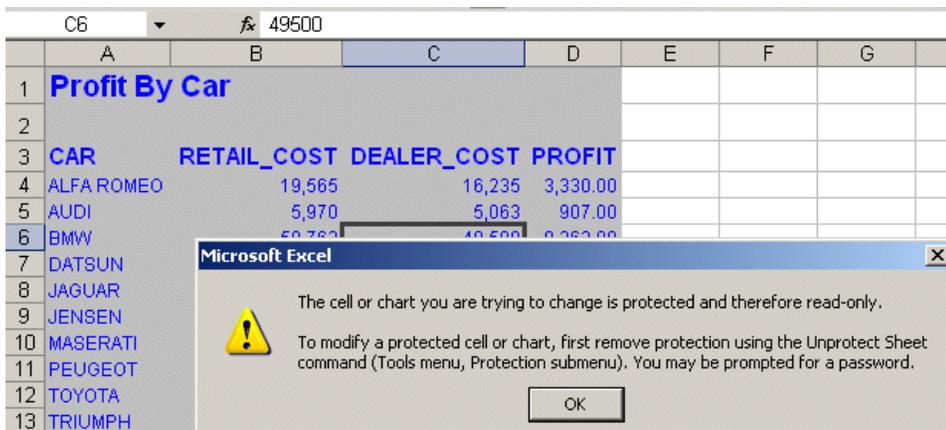
The following request locks the entire spreadsheet because the StyleSheet declarations include the following declaration:

```
TYPE=REPORT, PROTECTED=ON, $
```

The request is:

```
TABLE FILE CAR
HEADING
"Profit By Car "
" "
SUM RETAIL_COST AND DEALER_COST AND
COMPUTE PROFIT/D12.2 = RETAIL_COST - DEALER_COST;
BY CAR
ON TABLE SET PAGE-NUM OFF
ON TABLE PCHOLD AS EXLFORM1 FORMAT EXL2K
ON TABLE SET STYLE *
TYPE=REPORT, COLOR=BLUE, BACKCOLOR=SILVER, SIZE=9,$
TYPE=REPORT, PROTECTED=ON, $
TYPE=HEADING, STYLE=BOLD, SIZE=14, $
TYPE=TITLE, STYLE=BOLD, SIZE=11,$
ENDSTYLE
END
```

You cannot edit any value on the spreadsheet. Any attempt to do so displays a message that the sheet is protected:



Example: Locking a Single Column on an Excel Spreadsheet

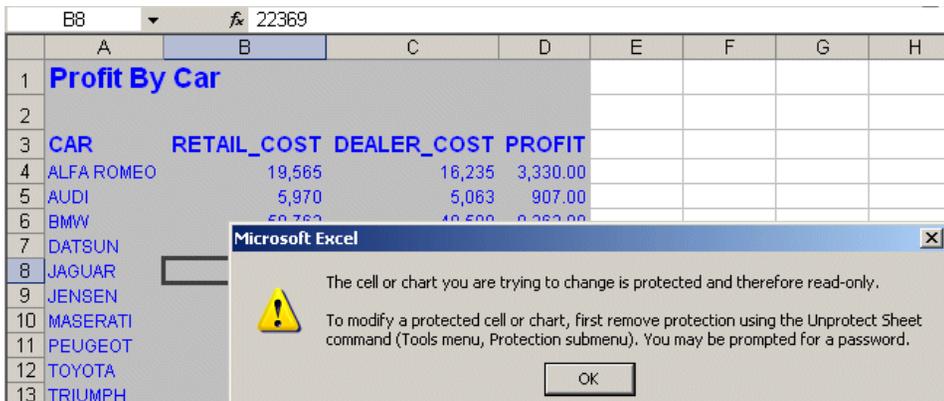
The following request locks the second column (RETAIL_COST) because the StyleSheet declarations include the following declarations

```
TYPE=REPORT, PROTECTED=ON, LOCKED=OFF, $
TYPE=DATA, COLUMN=2, LOCKED=ON, $
```

The request is:

```
TABLE FILE CAR
HEADING
"Profit By Car "
" "
SUM RETAIL_COST AND DEALER_COST AND
COMPUTE PROFIT/D12.2 = RETAIL_COST - DEALER_COST;
BY CAR
ON TABLE SET PAGE-NUM OFF
ON TABLE PCHOLD AS EXLFORM2 FORMAT EXL2K
ON TABLE SET STYLE *
TYPE=REPORT, COLOR=BLUE, BACKCOLOR=SILVER, SIZE=9,$
TYPE=REPORT, PROTECTED=ON, LOCKED=OFF,$
TYPE=HEADING, STYLE=BOLD, SIZE=14, $
TYPE=TITLE, STYLE=BOLD, SIZE=11,$
TYPE=DATA, COLUMN=2, LOCKED=ON,$
ENDSTYLE
END
```

You cannot edit any value in column 2, although you can edit values in other columns. Any attempt to edit a value in column 2 displays a message that the cells are protected:



Using the Excel 2000 Formula Option

How to:

Save a Report as FORMAT EXL2K FORMULA

Reference:

Translation Support for FORMAT EXL2K FORMULA

Since Excel users employ formulas to calculate the contents of spreadsheet cells, the EXL2K feature includes a HOLD option for generating correct native Excel formulas for all FOCUS summary operations, such as row and column totals and COMPUTE commands. The EXL2K StyleSheet-related features can also be applied with the FORMULA option.

When you display or save a tabular report request using EXL2K FORMULA, the resulting spreadsheet contains correct native Excel formulas that compute and display the results of all aggregation operations (such as row totals, column totals, subtotals, and calculated values) rather than the static values. Spreadsheets saved using the EXL2K FORMULA format are interactive, allowing for "what if" scenarios that immediately reflect any additions or modifications made to the data.

The EXL2K FORMULA format is supported for the following FOCUS TABLE options: ROWTOTAL, COLUMN-TOTAL, SUB-TOTAL, SUBTOTAL, SUMMARIZE, RECOMPUTE, and COMPUTE, and for calculations performed by functions.

EXL2K FORMULA is *not* supported with PivotTables (EXL2K PIVOT), with prefix operators, or with financial reports created with the Financial Modeling Language (FML).

Syntax: How to Save a Report as FORMAT EXL2K FORMULA

Add the following syntax to your request to include Excel formulas in your spreadsheet:

```
ON TABLE HOLD FORMAT EXL2K FORMULA
```

Example: Generating Native Excel Formulas for Column Totals

The following example illustrates the translation of a column total in a report request into an Excel formula when using format EXL2K FORMULA. Note that the formatting of the column total (TYPE=GRANDTOTAL) is retained in the Excel 2000 spreadsheet.

When you select the total in the report, the equation =SUM(B4:B7) appears in the formula bar, representing the column total as a sum of cell ranges.

```
TABLE FILE CENTORD
HEADING
"Projected Return By Region"
" "
SUM LINE_COGS AS 'RETURN'
BY REGION AS 'REGION'
ON TABLE COLUMN-TOTAL
ON TABLE HOLD AS EXL2K5 FORMAT EXL2K FORMULA
ON TABLE SET STYLE *
TYPE=REPORT, COLOR=BLUE, BACKCOLOR=SILVER, SIZE=9,$
TYPE=HEADING, STYLE=BOLD, SIZE=14,$
TYPE=TITLE, STYLE=BOLD+UNDERLINE, SIZE=10,$
TYPE=GRANDTOTAL, STYLE=BOLD,$
ENDSTYLE
END
```

The output is:

	A	B
1	Projected Return By Region	
2		
3	REGION	RETURN
4	EAST	233,492,476.00
5	NORTH	156,931,554.00
6	SOUTH	184,491,442.00
7	WEST	71,649,432.00
8	TOTAL	646,564,904.00

Example: Generating Native Excel Formulas for Row Totals

This request calculates totals for line price and quantity across regions. The row totals are represented as sums of cell ranges.

```
TABLE FILE CENTORD
HEADING
"Projected Line Cost Across Region"
" "
SUM LINEPRICE          AND QUANTITY
ACROSS REGION AS 'Region'
BY STORENAME
WHERE REGION EQ 'EAST' OR 'NORTH'
ON REGION ROW-TOTAL AS 'TOTAL'
ON TABLE COLUMN-TOTAL AS 'TOTAL'
ON TABLE HOLD AS EXL2K6 FORMAT EXL2K FORMULA
ON TABLE SET STYLE *
TYPE=REPORT, COLOR=BLUE, BACKCOLOR=SILVER, SIZE=9,$
TYPE=HEADING, STYLE=BOLD, SIZE=14,$
TYPE=TITLE, STYLE=BOLD, SIZE=11,$
TYPE=SUBTOTAL, STYLE=BOLD, $
TYPE=GRANDTOTAL, STYLE=BOLD, SIZE=11,$
TYPE=ACROSSTITLE, STYLE=BOLD, SIZE=11, JUSTIFY=LEFT,$
TYPE=ACROSSVALUE, STYLE=BOLD, SIZE=10, JUSTIFY=CENTER,$
ENDSTYLE
END
```

The output highlights the formula that calculates the row total in cell G11=C11+E11:

G11 =C11+E11							
	A	B	C	D	E	F	G
1	Projected Line Cost Across Region						
2							
3		Region					
4		EAST		NORTH		TOTAL	
5	Store	Line		Line		Line	
6	Name:	Total	Quantity:	Total	Quantity:	Total	Quantity:
7	eMart	\$48,833,656.98	196,084	\$114,336,646.64	457,064	\$163,170,303.62	653,148
8	Audio Expert	\$136,936,955.91	531,280	\$8,549,478.68	32,566	\$145,486,434.59	563,846
9	AV VideoTown	\$27,578,810.97	109,504	\$17,113,451.48	70,330	\$44,692,262.45	179,834
10	City Video	\$5,911,253.15	24,565	\$9,387,798.14	36,704	\$15,299,051.29	61,269
11	Consumer Merchandise	\$20,298,468.86	80,701	\$5,181,690.34	20,203	\$25,480,159.20	100,904
12	TV City	\$22,034,204.35	89,148	\$23,791,473.06	90,332	\$45,825,677.41	179,480
13	Web Sales	\$4,027,582.48	12,225			\$4,027,582.48	12,225
14	TOTAL	\$265,620,932.70	1,043,507	\$178,360,538.34	707,199	\$443,981,471.04	1,750,706

Example: Generating Native Excel Formulas for Calculated Values

This request totals the columns for line price and quantity and calculates the value of a field called LINECOST by multiplying the line price by the quantity.

The formula for the calculated values is generated by translating the internal form of the FOCUS expression (COMPUTE LINECOST/P24.2MC=QUANTITY*LINEPRICE;) into an Excel formula.

```
TABLE FILE CENTORD
ON TABLE SET PAGE-NUM OFF
SUM LINEPRICE AND QUANTITY AS 'Quantity'
COMPUTE LINECOST/P24.2MC = QUANTITY * LINEPRICE; AS 'Total Cost'
BY REGION AS 'Region'
HEADING
"Line Cost of Goods by Region"
" "
ON TABLE COLUMN-TOTAL
ON TABLE HOLD AS EXL2K7 FORMAT EXL2K FORMULA
ON TABLE SET STYLE *
TYPE=REPORT, COLOR=BLUE, BACKCOLOR=SILVER, SIZE=9,$
TYPE=HEADING, STYLE=BOLD, SIZE=14,$
TYPE=TITLE, STYLE=BOLD, SIZE=11,$
TYPE=GRANDTOTAL, STYLE=BOLD, SIZE=11,$
ENDSTYLE
END
```

The top line in the spreadsheet output highlights the formula.

	A	B	C	D
1	Line Cost of Goods by Region			
2				
3	Region	Line Total	Quantity	Total Cost
4	EAST	\$265,620,932.70	1,043,507	\$277,177,302,618,979.00
5	NORTH	\$178,360,538.34	707,199	\$126,136,394,353,510.00
6	SOUTH	\$208,533,625.34	827,871	\$172,638,940,943,851.00
7	WEST	\$81,507,927.63	325,673	\$26,544,931,315,045.00
8	TOTAL	\$734,023,024.01	2,904,250	\$602,497,569,231,385.00

Example: Generating Native Excel Formulas for Functions

The following illustrates how functions are translated to Excel 2000 reports. The function IMOD divides ACCTNUMBER by 1000 and returns the remainder to LAST3_ACCT. The Excel formula corresponds to this, =(MOD(\$C2,1000)).

```
TABLE FILE EMPLOYEE
PRINT ACCTNUMBER AS 'Account Number' AND COMPUTE
  LAST3_ACCT/I3L = IMOD(ACCTNUMBER, 1000, LAST3_ACCT);
BY LAST_NAME AS 'Last Name' BY FIRST_NAME AS 'First Name'
WHERE (ACCTNUMBER NE 00000000) AND (DEPARTMENT EQ 'MIS')
  ON TABLE HOLD AS EXL2K8 FORMAT EXL2K FORMULA
ON TABLE SET STYLE *
TYPE=TITLE, SIZE = 12, STYLE=BOLD,$
END
```

The output is:

D2 = =(MOD(\$C2,1000))				
	A	B	C	D
1	Last Name	First Name	Account Number	LAST3_ACCT
2	BLACKWOOD	ROSEMARIE	122850108	108
3	CROSS	BARBARA	163800144	144
4	GREENSPAN	MARY	150150302	302
5	JONES	DIANE	040950036	036
6	MCCOY	JOHN	109200096	096
7	SMITH	MARY	027300024	024

Reference: Translation Support for FORMAT EXL2K FORMULA

- ❑ All standard operators are supported. These include arithmetic operators, relational operators, string operators, IF/THEN/ELSE, and logical operators.
- ❑ The IS-MISSING operator is not supported.
- ❑ The following functions are supported: ABS, ARGLEN, ATODBL, BYTVAL, CHARGET, CTRAN, CTRFLD, DECODE DMOD, DOWK, DOWKI, DOWKL, DOWKLI, EDIT (1 argument variant only), EXP, EXPN, FMOD, HEXBYT, HHMMSS, IMOD, INT, LCWORD, LJUST, LOCASE, LOG, MAX, MIN, OVLAY, POSIT, RDUNIF, RANDOM, RJUST, SQRT, SUBSTR, TODAY, TODAYI, UPCASE, YM.
- ❑ If you use the COMPUTE command with an unsupported function, an error message appears.
- ❑ EXL2K FORMULA is not supported with the following FOCUS commands and phrases:

- ❑ DEFINE.
- ❑ OVER.
- ❑ NOPRINT. If your report contains a calculated value (generated by the COMPUTE command), all of the fields referenced by the calculated value must be displayed in the report.
- ❑ Multiple display (PRINT, LIST, SUM, and COUNT) commands.
- ❑ SEQUENCE.
- ❑ RECAP.
- ❑ Prefix operators.
- ❑ Formulas for ROW-TOTALs and COLUMN-TOTALs are represented as sums of cell ranges. For example, =SUM(G2:G10).
- ❑ A formula for a calculated value is generated by translating the internal form of the FOCUS expression into an Excel formula.
- ❑ Excel formulas are limited to 1024 characters. Therefore, the translation to an Excel formula will fail if a FOCUS total or the result of an expression requires more than 1024 bytes of Excel formula code. An error message will be generated.
- ❑ Conditional styling is based on the values in the original report. If the spreadsheet values are changed and the formulas are recomputed, the styling will not reflect the updated information.

Using the Excel 2000 PIVOT Option

How to:

Generate a PivotTable

Reference:

Usage Notes for PivotTable Requests

How TABLE Elements Appear in a Pivot Table

Effects of TABLE Syntax on PivotTable Requests

Content, Function and Origin of PivotTable Elements

The power of EXL2K format derives in large measure from its ability to take advantage of PivotTables. PivotTables are Microsoft Excel tools for analyzing complex data. They allow you to drag and drop data fields within a PivotTable spreadsheet, providing different views of the data, such as sorting across rows or columns. You can also create dimensional hierarchies by using the PAGEFIELDS option.

When you use FORMAT EXL2K PIVOT, two data streams are created:

- ❑ The first data stream is the PivotTable file containing the actual spreadsheet data. The PivotTable file (.xht) is an HTML file with embedded XML. The HTML file contains all the information that will be displayed in your browser.
- ❑ The second data stream is the PivotTable cache file (.xml). The PivotTable cache file is a metadata type file. It contains all the fields specified in the procedure and links internally to the PivotTable file. The PivotTable cache file can contain data fields called CACHEFIELDS, which populate the PivotTable toolbar, but do not initially display in the report. CACHEFIELDS can be dragged and dropped from the PivotTable toolbar into the PivotTable when required for analysis.

After creating FOCUS reports formatted as Excel PivotTables you must transfer both the XHT and XML files to Excel 2000 using FTP in ASCII mode or another transfer facility.

Syntax: **How to Generate a PivotTable**

```
ON TABLE HOLD FORMAT EXL2K PIVOT AS mypivot
```

where:

mypivot

Is a name you assign to the HOLD file.

Two files are generated with this syntax:

- ❑ MYPivot.XHT is the main PivotTable spreadsheet file that will be displayed in the browser or Excel window.
- ❑ MYPivot\$.XML is the Pivot Cache file.

Standard HOLD and SAVE syntax is supported for EXL2K PIVOT. The PivotTable cache file contains all the fields specified in the procedure and links internally to the PivotTable file. All available fields can be viewed in the PivotTable toolbar.

Reference: Usage Notes for PivotTable Requests

Keep these considerations in mind when preparing output for a PivotTable:

- ❑ Requests must include the display command PRINT. This is necessary to extract all data (each individual record) into the pivot cache file. The field following the PRINT command is designated the PivotTable Data field and must be numeric. Note that the SUM and COUNT commands are *not* supported for EXL2K PIVOT. Nevertheless, you can use this native PivotTable behavior in Excel once the data has been output to an Excel PivotTable.
- ❑ Fields used in a PivotTable cannot appear more than once in the report request. Each field can function in only one role in a FORMAT EXL2K PIVOT request. For example, a Page field cannot appear simultaneously as a column and a row field.
- ❑ Styling is based on the initial state of your report and is retained only on a limited basis when you pivot the output.
- ❑ NODATA fields appear as blank cells in the PivotTable.
- ❑ You can include column, row, grand totals, and subtotals in a PivotTable. If no totals are specified in a request, no totals will display in Excel 2000.

Example: Using the EXL2K PIVOT Option

This simple example shows how to populate and generate PivotTables:

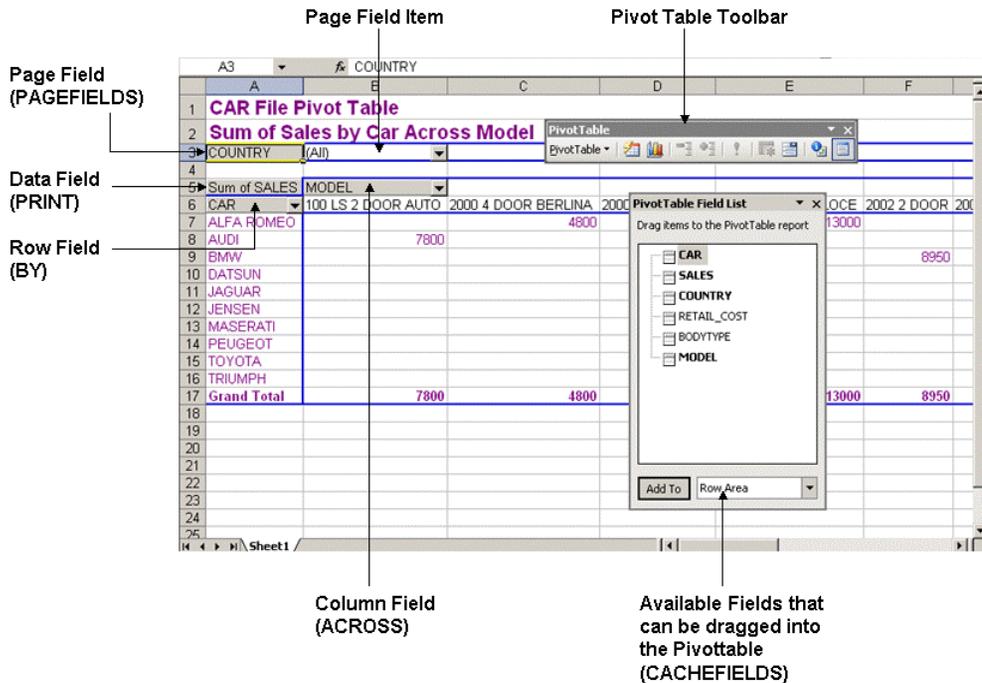
```
TABLE FILE CENTINV
HEADING
"CENTINV File PivotTable"
"Sum of Price by Product Across Category"
PRINT PRICE
BY PROD_NUM
ACROSS PRODCAT
ON TABLE COLUMN-TOTAL
  ON TABLE HOLD AS EXL2K9 FORMAT EXL2K PIVOT
  PAGEFIELDS PRODNAME
  CACHEFIELDS COST QTY_IN_STOCK
ON TABLE SET STYLE *
TYPE=HEADING, LINE=1, FONT='ARIAL', COLOR=PURPLE, SIZE=16, STYLE=BOLD,$
TYPE=HEADING, LINE=2, FONT='ARIAL', COLOR=PURPLE, SIZE=12, STYLE=BOLD,$
TYPE=DATA, FONT='ARIAL', COLOR=PURPLE,$
TYPE=GRANDTOTAL, FONT='ARIAL', COLOR=PURPLE, SIZE=12, STYLE=BOLD,$
ENDSTYLE
END
```

The output is:

Product Number	Camcorders	Cameras	CD Players	Digital Cameras	Digital Video Cameras	Digital Video Recorders	VCRs
1004							179.00
1006							
1008							
1010	999.00						
1012	899.00						
1014		249.00					
1016		279.00					
1018	399.00						
1020	319.00						
1022	399.00						
1024	349.00						
1026		129.00					
1028		109.00					
1030			169.00				
1032							
1034							9.00
1036							9.00
Grand Total	3,364.00	766.00	169.00	89.00	598.00	798.00	179.00

Reference: How TABLE Elements Appear in a Pivot Table

The PivotTable is generated by the PRINT command in combination with the BY, ACROSS, PAGEFIELDS, and CACHEFIELDS phrases. It contains all options used to design and format the report, as well as fields specified in the PIVOT request. Fields can be dragged into the report from the toolbar. The following graphic and summary table depicts PivotTable output with major elements and associated FOCUS syntax.



Reference: Effects of TABLE Syntax on PivotTable Requests

This table summarizes the effects of TABLE request syntax elements within EXL2K PIVOT operations. You must include at least one sort field or a PAGEFIELD to have a valid Pivot Table request.

Syntax Element	Usage	Effect on PivotTable
PRINT	Required	Designates the data field in a PivotTable.
BY	Optional	Designates row field in a PivotTable.

Syntax Element	Usage	Effect on PivotTable
ACROSS	Optional	Designates a column field in a PivotTable.
CACHEFIELDS	Optional	Places fields in the Pivot cache file and makes them available from the Pivot toolbar.
PAGEFIELDS	Optional	Designates a Page field in a PivotTable.

Reference: Content, Function and Origin of PivotTable Elements

PivotTable Element	Contains...	Function	Generating Syntax
Page field	Field that controls view of the entire page (worksheet).	A filtering mechanism to conduct a high level sort.	PAGEFIELDS phrase
Page field item	The value for a page field item appears in a drop-down list.	Selecting a page field item summarizes data for the entire report.	PAGEFIELDS phrase
Data field	Numeric data that is available to be summarized	Holds data available to be summarized	PRINT command
Column field	Horizontal sort data	Sorts data horizontally.	ACROSS command
Row field	Vertical sort data	Sorts data vertically.	BY command

Designating CACHEFIELDS in PivotTables

Reference:

Usage Notes for Specifying CACHEFIELDS

By including a CACHEFIELDS phrase in your EXL2K request, you can add fields to the pivot cache not initially displayed in the report. The cache file enables you to add available fields from the PivotTable toolbar into the body of the PivotTable by dragging and dropping. You can also remove fields from the PivotTable by dragging and dropping them outside the report. In either case, you can very quickly vary your data views.

The CACHEFIELDS phrase is optional; you can always generate a PivotTable without one.

Reference: Usage Notes for Specifying CACHEFIELDS

Fields designated as CACHEFIELDS must immediately follow the PIVOT keyword in the ON TABLE HOLD FORMAT EXL2K PIVOT syntax or follow a PAGEFIELDS phrase. A CACHEFIELD cannot be designated elsewhere in the request. Lists of CACHEFIELDS are terminated by the same keywords that terminate normal report requests, such as END or another ON phrase.

Example: Using CACHEFIELDS With EXL2K PIVOT

This example shows how to specify CACHEFIELDS to populate the PivotTable toolbar.

```
TABLE FILE CENTINV
HEADING
"PivotTable with CACHEFIELDS"
"Sum of Price by Product Across Category"
PRINT PRICE
BY PRODCAT BY PRODTYPE
ON PRODCAT SUB-TOTAL
  ON TABLE HOLD AS EXL2K10 FORMAT EXL2K PIVOT
  CACHEFIELDS COST PRODNAME
ON TABLE SET STYLE *
TYPE=DATA, COLUMN=PRODCAT, COLOR=RED,$
TYPE=HEADING, COLOR=BLUE, STYLE=BOLD, SIZE=14,$
ENDSTYLE
END
```

The output is:

Sum of Price:	Product Type:	Total
Camcorders	Analog	1,466.00
	Digital	1,898.00
Camcorders Total		3,364.00
Cameras	Analog	238.00
	Digital	528.00
Cameras Total		766.00
CD Players	Digital	169.00
CD Players Total		169.00
Digital Tape Recorders	Digital	89.00
Digital Tape Recorders Total		89.00
DVD	Digital	598.00
DVD Total		598.00
PDA Devices	Digital	798.00
PDA Devices Total		798.00
VCRs	Analog	179.00
VCRs Total		179.00
Grand Total		5,963.00

Designating PAGEFIELDS in PivotTables

You can specify fields in the procedure as Excel 2000 page fields. Page fields filter the data for the field value specified. Through PivotTable functionality, you can select a single value from the page field drop-down menu (also called a page field item) to display only data associated with that selection. For example, in a report showing international car sales data, if you specify COUNTRY as your page field and JAPAN as the page field item, you will display only sales data for Japanese cars. If you then select ENGLAND, you will see the data for JAGUAR and TRIUMPH.

A page field can act as the sort field. Valid PivotTables can be generated without specifying a PAGEFIELD if sorting is handled by either a BY or ACROSS phrase. However, if the request contains neither a BY or ACROSS phrase, a PAGEFIELD must be included.

Note:

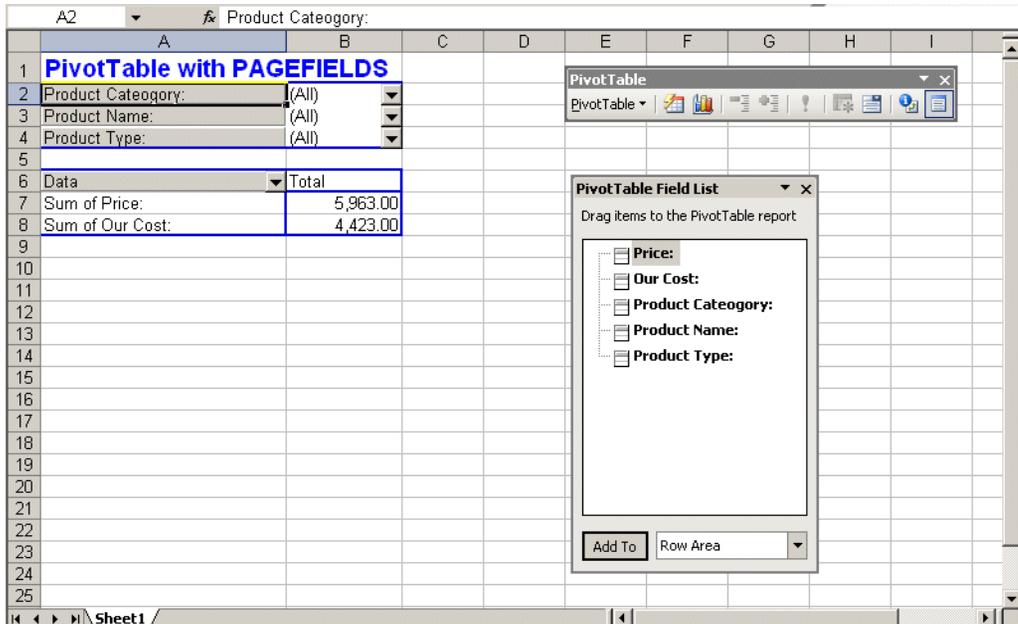
- ❑ A field specified as a PAGEFIELD cannot be designated anywhere else in the request.
- ❑ Because Excel is case insensitive, the values of the PAGEFIELD must not contain duplicate values in different cases. For example, "JONES" and "jones" are considered equal in Excel. Use the UPCASE and LOCASE functions to convert mixed-case values.

Example: Using FORMAT EXL2K PIVOT With PAGEFIELDS

This example illustrates the use of PAGEFIELDS syntax to make three fields available in the PivotTable toolbar.

```
TABLE FILE CENTINV
HEADING
"PivotTable with PAGEFIELDS"
PRINT PRICE COST
  ON TABLE HOLD AS EXL2K11 FORMAT EXL2K PIVOT
  PAGEFIELDS PRODCAT PRODNAME PRODTYPE
ON TABLE SET STYLE *
TYPE=DATA, COLUMN=PRODCAT, COLOR=RED,$
TYPE=HEADING, COLOR=BLUE, STYLE=BOLD, SIZE=14,$
ENDSTYLE
END
```

This output is:



Excel Named Ranges

How to:

Use Excel Named Ranges

Reference:

Rules for Excel Named Ranges

Support for Excel Named Ranges

An Excel Named Range is a name assigned to a specific group of cells within an Excel worksheet that can be easily referenced by FOCUS applications. The FOCUS StyleSheet language facilitates the generation of Named Ranges.

The use of Excel Named Ranges provides many benefits, including the following:

- ❑ Provides advantages over static cell references, including the ability of named range data areas to expand to include new data added during scheduled workbook updates.
- ❑ Enables easy setup of Excel worksheets, created by FOCUS applications, as an ODBC (Open Database Connectivity) data source.
- ❑ Provides accurate, consistent data feeds to advanced Excel worksheet applications, which eliminates manual activities that tend to result in errors.
- ❑ Simplifies the process of referencing data in multiple worksheets. This is especially useful when named ranges are added to the output of an Excel Template report.

Syntax: **How to Use Excel Named Ranges**

To create Excel Named Ranges, use

`TYPE=type, IN-RANGES=rangename, $`

where:

type

Identifies the FOCUS report component to be included in the range. Normally, both of the following are used together:

DATA adds the DATA element of the report to the named range (excludes heading, footing, and column titles).

TITLE adds the TITLE element of the report to the named range (includes all column titles).

Note: Multiple elements can be added to the same named range.

rangename

Is the name assigned to the output in the Excel workbook your application is creating, and is also the name that will be referenced by other FOCUS applications.

Example: Using Excel Named Ranges

This example creates one report in one worksheet of an Excel workbook. The code specific to Excel Named Ranges appears in bold in the following syntax.

```
TABLE FILE GGSALES
PRINT
    PRODUCT
    DATE
    UNITS
BY REGION
BY DOLLARS
ON TABLE SET PAGE-NUM OFF
ON TABLE SET BYDISPLAY ON
ON TABLE NOTOTAL
ON TABLE HOLD FORMAT EXL2K
ON TABLE SET STYLE *
    UNITS=IN, SQUEEZE=ON, ORIENTATION=PORTRAIT, $
TYPE=REPORT, FONT='ARIAL', SIZE=9, COLOR='BLACK', BACKCOLOR='NONE',
    STYLE=NORMAL, $
TYPE=DATA, IN-RANGES='RegionalSales', $
TYPE=TITLE, STYLE=BOLD, IN-RANGES='RegionalSales', $
ENDSTYLE
END
```

The Excel output is:

RegionalSales		Region				
	A	B	C	D	E	F
1	Region	Dollar Sales	Product	Date	Unit Sales	
2	Midwest	748	Thermos	1996/08/01	68	
3	Midwest	944	Mug	1997/08/01	86	
4	Midwest	980	Thermos	1996/07/01	89	
5	Midwest	1010	Scone	1996/09/01	101	
6	Midwest	1060	Thermos	1996/02/01	96	
7	Midwest	1069	Mug	1996/11/01	89	
8	Midwest	1100	Croissant	1996/11/01	100	
9	Midwest	1111	Espresso	1997/08/01	101	
10	Midwest	1111	Latte	1996/11/01	101	
11	Midwest	1230	Espresso	1996/07/01	123	
12	Midwest	1247	Coffee Grinder	1996/06/01	125	

The name assigned to this Excel Named Range is RegionalSales. If additional rows of data are added, or columns of data are inserted, the named range will stretch to contain both new and existing data.

Reference: Rules for Excel Named Ranges

- ❑ The Excel data area associated with a named range must be continuous and cannot contain any breaks in the data. Examples of report components containing breaks in the data that cannot be part of a named range include SUBHEAD and SUBFOOT.
- ❑ It is recommended that you use ON TABLE SET BYDISPLAY ON. This activates the option to display repeated sort values, which produces continuous output with no breaks in the data.
- ❑ Two different worksheets from the same workbook cannot have the same range name.
- ❑ When creating Compound Excel reports (multiple TABLE requests output to the same Excel workbook), each report must have a unique range name that is stored at the workbook level.

Reference: Support for Excel Named Ranges

Excel Named Ranges are supported for the following Excel formats:

[EXL2K](#), [EXL2K FORMULA](#), [EXL2K TEMPLATE](#)

Excel Named Ranges are not supported for the following Excel formats:

[EXL2K BYTOC](#), [EXCEL PIVOT](#)

Excel Named Ranges are not supported with any report syntax that produces discontinuous data or uses columnar references that span multiple columns, which includes the following:

[ACROSSCOLUMN](#), [RECAP](#), [RECOMPUTE](#), [SUBHEAD](#), [SUBFOOT](#), [SUBTOTAL](#), [SUB-TOTAL](#)

Identifying Null Values in Excel 2000

How to:

Identify Null Values in EXL2K Report Output

When a report is formatted as EXL2K, and null values are retrieved for one or more fields, blank spaces are displayed by default in each cell of the report output for the empty (null) fields. This behavior is the result of SET EMPTYCELLS ON being set by default in the background of all EXL2K reports. If you want to identify null values with something other than blank spaces, a character string can be used to populate all empty fields in a report.

Syntax: **How to Identify Null Values in EXL2K Report Output**

Outside of a report request, use the following syntax

```
SET NODATA = character_string  
SET EMPTYCELLS = [ON|OFF]
```

In a report request, use the following syntax

```
ON TABLE SET NODATA character_string  
ON TABLE SET EMPTYCELLS [ON|OFF]
```

where:

character_string

Is the string of characters displayed in the cells of the report for each field where null values are retrieved from the database. The maximum number of characters is 11. If the number of characters in the string exceeds the length of the output field, the additional characters will not be displayed. If special characters are used, the string must be enclosed in single quotes. SET EMPTYCELLS OFF must also be specified to make the SET NODATA command effective.

ON

Indicates that empty spaces are displayed in the cells of the report for each field where null values are retrieved from the database. ON is the default.

OFF

Indicates that zeros, or the character string specified with the SET NODATA command, will be displayed in the cells of the report for each field where null values are retrieved from the database. OFF must be specified when using SET NODATA.

Example: **Identifying Null Values in EXL2K Report Output**

The following syntax utilizes the default behavior of ON TABLE SET EMPTYCELLS ON, which is set in the background:

```
TABLE FILE CAR  
SUM SALES BY COUNTRY ACROSS SEATS  
ON TABLE HOLD FORMAT EXL2K  
ON TABLE SET STYLE *  
TYPE = ACROSSTITLE, STYLE=BOLD,$TYPE = TITLE, STYLE = BOLD,$  
ENDSTYLE  
END
```

The following output displays empty spaces in the cells of the report for each field where null values are retrieved from the database:

	A	B	C	D	E
1		SEATS			
2		2	4	5	
3	COUNTRY				
4	ENGLAND	0	0	12000	
5	FRANCE			0	
6	ITALY	25400	4800		
7	JAPAN		78030		
8	W GERMANY		8900	79290	
9					

The following syntax utilizes the SET NODATA command:

```
TABLE FILE CAR
SUM SALES BY COUNTRY ACROSS SEATS
ON TABLE SET NODATA 'n/a'
ON TABLE SET EMPTYCELLS OFF
ON TABLE HOLD FORMAT EXL2K
ON TABLE SET STYLE *
TYPE = ACROSSTITLE, STYLE=BOLD,$TYPE = TITLE, STYLE = BOLD,$
ENDSTYLE
END
```

Note: If you do not add SET EMPTYCELLS OFF, the SET NODATA command will be ignored.

The following output displays 'n/a' in the cells of the report for each field where null values are retrieved from the database:

	A	B	C	D	E
1		SEATS			
2		2	4	5	
3	COUNTRY				
4	ENGLAND	0	0	12000	
5	FRANCE	n/a	n/a	0	
6	ITALY	25400	4800	n/a	
7	JAPAN	n/a	78030	n/a	
8	W GERMANY	n/a	8900	79290	
9					

The following syntax turns off the default SET EMPTYCELLS behavior and does not use SET NODATA, which makes it impossible to distinguish null values from zero quantities:

```
TABLE FILE CAR
SUM SALES BY COUNTRY ACROSS SEATS
ON TABLE SET EMPTYCELLS OFF
ON TABLE HOLD FORMAT EXL2K
ON TABLE SET STYLE *
TYPE = ACROSSTITLE, STYLE=BOLD,$TYPE = TITLE, STYLE = BOLD,$
ENDSTYLE
END
```

The following output displays zeros in the cells of the report for each field where either null values are retrieved from the database or the quantity is zero:

	A1				
	A	B	C	D	E
1		SEATS			
2		2	4	5	
3	COUNTRY				
4	ENGLAND	0	0	12000	
5	FRANCE	0	0	0	
6	ITALY	25400	4800	0	
7	JAPAN	0	78030	0	
8	W GERMANY	0	8900	79290	
9					

Excel Table of Contents

How to:
Use the Excel Table of Contents Feature

Reference:
How to Name Worksheets
Limitations of TOC Reports

Excel Table of Contents (TOC) is a feature that enables you to generate a multiple worksheet report in which a separate worksheet is generated for each value of the first BY field in the FOCUS report.

Note: This feature can be used only with Excel 2002 or higher releases because it requires the Web Archive file format, which was not available in Excel 2000 and earlier releases.

Syntax: How to Use the Excel Table of Contents Feature

Only a single BY field is allowed in an EXL2K Table of Contents report.

```
ON TABLE HOLD FORMAT EXL2K BYTOC
```

Since only one level of TOC is allowed for EXL2K reports, the optional number following the BYTOC keyword can only be 1.

The SET COMPOUND syntax, which precedes the TABLE command, may also be used to specify that a TOC be created:

```
SET COMPOUND=BYTOC
```

Since a TOC report is burst into worksheets according to the value of the first BY field in the report, the report must contain at least one BY field. The bursting field may be a NOPRINT field.

Reference: How to Name Worksheets

- ❑ The worksheet tab names are the BY field values that correspond to the data on the current worksheet. If the user specifies the TITLETEXT keyword in the stylesheet, it will be ignored.
- ❑ Excel limits the length of worksheet titles to 31 characters. The following special characters cannot be used: ':', '?', '*', and '/'.

Reference: Limitations of TOC Reports

- ❑ A TOC report cannot be embedded in a compound report.
- ❑ A TOC report cannot be a pivot table report.

Example: Creating a Simple TOC Report

```
SET COMPOUND=BYTOC
TABLE FILE CAR
PRINT SALES BY COUNTRY NOPRINT BY CAR
ON TABLE HOLD FORMAT EXL2K
ON TABLE SET STYLE *
type=report, style=bold, color=yellow, backcolor=black, $
type=data, backcolor=red, $type=data, column=car, color=blue,
backcolor=yellow, $
END
```


Note: Excel 2002 (Office XP) or higher must be installed. This feature will not work with earlier versions of Excel since they do not support the Web Archive file format.

Reference: Guidelines for Using the OPEN, CLOSE, and NOBREAK Keywords and SET COMPOUND

As with PDF, the keywords OPEN, CLOSE, and NOBREAK are used to control Excel compound reports. They can be specified with the HOLD command or with a separate SET COMPOUND command.

- ❑ OPEN is used on the first report of a sequence of component reports to specify that a compound report be produced.
- ❑ CLOSE is used to designate the last report in a compound report.
- ❑ NOBREAK specifies that the next report be placed on the same worksheet as the current report. If it is not present, the default behavior is to place the next report on a separate worksheet.

NOBREAK may appear with OPEN on the first report, or alone on a report between the first and last reports. (Using CLOSE is irrelevant, since it refers to the placement of the next report, and no report follows the final report on which CLOSE appears.)

- ❑ When used with the HOLD syntax, the compound report keywords OPEN, CLOSE, and NOBREAK must appear immediately after FORMAT EXL2K, and before any additional keywords, such as FORMULA or PIVOT. For example, you can specify:
 - ❑ `ON TABLE HOLD FORMAT EXL2K OPEN`
 - ❑ `ON TABLE HOLD AS MYHOLD FORMAT EXL2K OPEN NOBREAK`
 - ❑ `ON TABLE HOLD FORMAT EXL2K NOBREAK FORMULA`
 - ❑ `ON TABLE HOLD FORMAT EXL2K CLOSE PIVOT PAGEFIELDS COUNTRY`
- ❑ As with PDF compound reports, compound report keywords can be alternatively specified using SET COMPOUND:
 - ❑ `SET COMPOUND = OPEN`
 - ❑ `SET COMPOUND = 'OPEN NOBREAK'`
 - ❑ `SET COMPOUND = NOBREAK`
 - ❑ `SET COMPOUND = CLOSE`

Reference: Guidelines for Producing Excel Compound Reports

- ❑ **Pivot Tables and NOBREAK.** Pivot Table Reports may appear in compound reports, but they may not be combined with another report on the same worksheet using NOBREAK.

- ❑ **Naming of Worksheets.** The default worksheet tab names will be Sheet1, Sheet2, and so on. You have the option to specify a different worksheet tab name by using the TITLETEXT keyword in the stylesheet. For example:

```
TYPE=REPORT, TITLETEXT='Summary Report', $
```

Excel limits the length of worksheet titles to 31 characters. The following special characters cannot be used: ':', '?', '*', and '/'.

- ❑ **File Names and Formats.** The output file name (AS name, or HOLD by default) is obtained from the first report of the compound report (the report with the OPEN keyword). Output file names on subsequent reports are ignored.

The HOLD FORMAT syntax used in the first component report in a compound report applies to all subsequent reports in the compound report, regardless of their format.

- ❑ **NOBREAK Behavior.** When NOBREAK is specified, the following report appears on the row immediately after the last row of the report with the NOBREAK. If additional spacing is required between the reports, a FOOTING or an ON TABLE SUBFOOT can be placed on the report with the NOBREAK, or a HEADING or an ON TABLE SUBHEAD can be placed on the following report. This allows the most flexibility, since if blank rows were added by default there would be no way to remove them.

Example: Creating a Simple Compound Report Using EXL2K

```
SET PAGE-NUM=OFF
TABLE FILE CAR
HEADING
"Sales Report"
" "
SUM SALES
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Sales Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD AS EX1 FORMAT EXL2K OPEN
END
```

```
TABLE FILE CAR
HEADING
"Inventory Report"
" "
SUM RC
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Inv. Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD FORMAT EXL2K
END
```

```
TABLE FILE CAR
HEADING
"Cost of Goods Sold Report"
" "
SUM DC
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Cost Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD FORMAT EXL2K CLOSE
END
```

The output for each tab in the Excel worksheet is:

A1 fx Sales Report			
	A	B	C
1	Sales Report		
2			
3	COUNTRY	SALES	
4	ENGLAND	12000	
5	FRANCE	0	
6	ITALY	30200	
7	JAPAN	78030	
8	W GERMANY	88190	
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			

Navigation: Sales Rpt / Inv. Rpt / Cost Rpt /

A1 fx Inventory Report			
	A	B	C
1	Inventory Report		
2			
3	COUNTRY	RETAIL_COST	
4	ENGLAND	45,319	
5	FRANCE	5,610	
6	ITALY	51,065	
7	JAPAN	6,478	
8	W GERMANY	64,732	
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			

Navigation: Sales Rpt / **Inv. Rpt** / Cost Rpt /

A1		Cost of Goods Sold Report	
A		B	
1	Cost of Goods Sold Report		
2			
3	COUNTRY	DEALER_COST	
4	ENGLAND	37,853	
5	FRANCE	4,631	
6	ITALY	41,235	
7	JAPAN	5,512	
8	W GERMANY	54,563	
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			

Example: Creating a Compound Report With Pivot Tables and Formulas

```

SET PAGE-NUM=OFF
TABLE FILE CAR
HEADING
"Sales Report"
" "
PRINT RCOST
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Sales Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD AS PIV1 FORMAT EXL2K OPEN
END

```

```
TABLE FILE CAR
HEADING
"Inventory Report"
" "
PRINT SALES
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Inv. Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD AS PPPP FORMAT EXL2K PIVOT
PAGEFIELDS TYPE SEATS
CACHEFIELDS MODEL MPG RPM
END

TABLE FILE CAR
SUM RCOST
BY COUNTRY BY CAR BY MODEL BY TYPE BY SEATS SUMMARIZE
ON MODEL SUB-TOTAL
ON TABLE HOLD AS XFOCB FORMAT EXL2K FORMULA
END

TABLE FILE CAR
HEADING
"Cost of Goods Sold Report"
" "
PRINT DCOST
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Cost Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD AS ONE FORMAT EXL2K CLOSE PIVOT
PAGEFIELDS RCOST
CACHEFIELDS MODEL TYPE SALES ACCEL SEATS
END
```

The output for each tab in the Excel worksheet is:

Sales Report	
COUNTRY	RETAIL_COST
ENGLAND	8,878
	13,491
	17,850
	5,100
FRANCE	5,610
ITALY	5,925
	6,820
	6,820
	31,500
JAPAN	3,139
	3,339
W GERMANY	5,970
	5,940
	6,355
	13,752
	14,123
	9,097
	9,495

Inventory Report	
BODYTYPE	(All)
SEATS	(All)
Sum of SALES	
COUNTRY	Total
ENGLAND	12000
FRANCE	0
ITALY	30200
JAPAN	78030
W GERMANY	88190

	A	B	C	D	E	F	G
1	COUNTRY	CAR	MODEL	BODYTYPE	SEATS	RETAIL_COST	
2	ENGLAND	JAGUAR	V12XKE AUTO	CONVERTIBLE	2	8,878	
3	*TOTAL SEATS 2					8,878	
4	*TOTAL BODYTYPE CONVERTIBLE					8,878	
5	*TOTAL MODEL V12XKE AUTO					8,878	
6			XJ12L AUTO	SEDAN	5	13,491	
7	*TOTAL SEATS 5					13,491	
8	*TOTAL BODYTYPE SEDAN					13,491	
9	*TOTAL MODEL XJ12L AUTO					13,491	
10	*TOTAL CAR JAGUAR					22,369	
11		JENSEN	INTERCEPTOR III	SEDAN	4	17,850	
12	*TOTAL SEATS 4					17,850	
13	*TOTAL BODYTYPE SEDAN					17,850	
14	*TOTAL MODEL INTERCEPTOR III					17,850	
15	*TOTAL CAR JENSEN					17,850	
16		TRIUMPH	TR7	HARDTOP	2	5,100	
17	*TOTAL SEATS 2					5,100	
18	*TOTAL BODYTYPE HARDTOP					5,100	
19	*TOTAL MODEL TR7					5,100	
20	*TOTAL CAR TRIUMPH					5,100	
21	*TOTAL COUNTRY ENGLAND					45,319	
22	FRANCE	PEUGEOT	504 4 DOOR	SEDAN	5	5,610	
23	*TOTAL SEATS 5					5,610	
24	*TOTAL BODYTYPE SEDAN					5,610	

A1 Cost of Goods Sold Report

	A	B
1	Cost of Goods Sold Report	
2		
3	RETAIL_COST	(All) ▾
4		
5	Sum of DEALER_COST	
6	COUNTRY ▾	Total
7	ENGLAND	37,853
8	FRANCE	4,631
9	ITALY	41,235
10	JAPAN	5,512
11	W GERMANY	54,563
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		

Example: Creating a Compound Report Using NOBREAK

In this example, the first two reports are on the first worksheet, and the last two reports are on the second worksheet, since NOBREAK appears on both the first and third reports.

```
TABLE FILE CAR
HEADING
"Report 1: England"
PRINT DCOST BY COUNTRY BY CAR
IF COUNTRY EQ ENGLAND
ON TABLE HOLD FORMAT EXL2K OPEN NOBREAK
ON TABLE SET STYLE *
type=report, color=red, $
type=data, bgcolor=yellow, $
type=heading, color=blue, $
END

TABLE FILE CAR
HEADING
" "
" "
"Report 2: France"
PRINT RCOST BY COUNTRY
IF COUNTRY EQ FRANCE
ACROSS SEATS
ON TABLE HOLD FORMAT EXL2K
ON TABLE SET STYLE *
type=report, color=lime, bgcolor=fuschia, style=bold, $
type=title, color=black, style=bold+italic, $
type=heading, bgcolor=black, $
END

TABLE FILE CAR
FOOTING
"Report 3 - All"
PRINT SALES BY COUNTRY BY CAR
ON TABLE HOLD FORMAT EXL2K NOBREAK
ON TABLE SET STYLE *
type=report, bgcolor=yellow, style=bold, $
type=title, color=blue, $
type=footing, bgcolor=aqua, $
END
```

```

TABLE FILE CAR
HEADING
" "
" "
"Report 4"
PRINT SALES BY COUNTRY BY CAR
ON TABLE HOLD FORMAT EXL2K CLOSE
ON TABLE SET STYLE *
type=report, color=yellow, bgcolor=black, style=bold, $
END
    
```

The output is:

Report 1: England		
A	B	C
Report 1: England		
COUNTRY	CAR	DEALER_COST
ENGLAND	JAGUAR	7,427
		11,194
	JENSEN	14,940
	TRIUMPH	4,292
Report 2: France		
	SEATS	
	\$	
COUNTRY		
FRANCE	5,610	

COUNTRY	CAR	SALES
ENGLAND	JAGUAR	0
		12000
	JENSEN	0
	TRIUMPH	0
FRANCE	PEUGEOT	0
ITALY	ALFA ROMEO	4800
		12400
		13000
	MASERATI	0
JAPAN	DATSUN	43000
	TOYOTA	35030
W GERMANY	AUDI	7800
	BMW	8950
		8900
		14000
		18940
		14000
		15600
Report 3 - All		
Report 4		
COUNTRY	CAR	SALES
ENGLAND	JAGUAR	0
		12000
	JENSEN	0
	TRIUMPH	0
FRANCE	PEUGEOT	0
ITALY	ALFA ROMEO	4800
		12400
		13000
	MASERATI	0
JAPAN	DATSUN	43000
	TOYOTA	35030
W GERMANY	AUDI	7800
	BMW	8950
		8900
		14000
		18940
		14000
		15600

Transferring Excel 2000 Formatted Files Using FTP

Reference:

Important Considerations for Transferring EXL2K-generated Files

After creating an Excel 2000 formatted file, you must transfer it from the mainframe to your PC to view and use it. The following illustrates the process of using FTP in Microsoft Windows to retrieve the files from the mainframe:

```
C:\temp\work>ftp ibimvs
Connected to ibimvs.ibi.com.
220-FTPD1 IBM FTP CS V2R10 at IBIMVS.IBI.COM, 15:24:50 on 2003-11-06.
220 Connection will close if idle for more than 5 minutes.
User (ibimvs.ibi.com:(none)): userid1
331 Send password please.
Password:
230 USERID1 is logged on. Working directory is "USERID1.".
ftp> get pivotmvs.xht pivot.xht
200 Port request OK.
125 Sending data set USERID1.PIVOTMVS.XHT
250 Transfer completed successfully.
ftp: 8387 bytes received in 0.14Seconds 59.48Kbytes/sec.
ftp> get pivotmvs.xml pivot.xml
200 Port request OK.
125 Sending data set USERID1.PIVOTMVS.XML
250 Transfer completed successfully.
ftp: 1940 bytes received in 0.16Seconds 12.44Kbytes/sec.
ftp> by
221 Quit command received. Goodbye.
```

Reference: Important Considerations for Transferring EXL2K-generated Files

- ❑ You must transfer the file as an ASCII file.
- ❑ Whatever name you give the file ("HOLD" or an "asname" assigned with a HOLD AS phrase) must be kept for the transferred file. The HOLD EXL2K PIVOT operation actually produces two files; an "asname.XHT" data file and an "asname\$.XML" PivotTable file.
- ❑ Before the FTP operation on z/OS, issue a DYNAM FREE for both the "asname" and the "asname\$" ddnames.

Creating Styled Excel 97 Files

How to:

Create a Styled Excel 97 File

Reference:

Limitations for FORMAT EXL97

EXL97 is an HTML-based HOLD format for generating formatted Excel 97 spreadsheets. EXL97 is a full StyleSheet driver for accurately rendering all report elements (such as headings and subtotals, for example) as well as applying StyleSheet syntax (such as conditional styling). You must have Microsoft Excel 97 or higher installed on your computer to display an Excel 97 report.

While Excel 97 is fully compatible with Excel 2000 and Excel 2002, we strongly recommend upgrading to Excel 2000 to exploit its broader range of features and future Excel enhancements, which will primarily be made to the EXL2K format. See [Limitations for FORMAT EXL97](#) on page 687.

Syntax: **How to Create a Styled Excel 97 File**

To produce an Excel 97 spreadsheet, create a FOCUS report using the Excel 97 HOLD option and then transfer the output file to your browser and open it in Excel 97. The HOLD syntax is

```
[ON TABLE] HOLD [AS filename] FORMAT EXL97
```

where:

`EXL97`

Creates an Excel-formatted HTML file, with an extension of .HTM, which may include styling based on FOCUS StyleSheet features. The MIME type assigned automatically designates Excel as the active application for this file type. Before you can see or work with this file you must transfer it to your PC.

Example: Creating an EXL97 Output File

The following example shows how to create a report in EXL97 format based on the contents of CENTORD, with conditional styling:

```
TABLE FILE CENTORD
HEADING
"Order Revenue"
"Styled Report in Excel 97"
" "
SUM ORDER_DATE LINEPRICE AS 'Order,Total:'
BY HIGHEST 10 ORDER_NUM
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT EXL97
ON TABLE SET STYLE *
TYPE=HEADING, COLOR=NAVY, SIZE=10, $
TYPE=HEADING, LINE=2, COLOR=RED, $
TYPE=DATA, BACKCOLOR=AQUA, STYLE=BOLD, WHEN=LINEPRICE GT 200000, $
TYPE=TITLE, STYLE=BOLD, $
END
```

The output is:

	A	B	C
1	Order Revenue		
2	Styled Report in Excel 97		
3			
4	Order	Date	Order
5	Number:	Of Order:	Total:
6	94710	1/2/2001	\$406,964.24
7	94680	1/2/2001	\$421,916.60
8	94670	1/2/2001	\$513,868.76
9	94550	1/2/2001	\$496,323.64
10	94530	1/2/2001	\$3,472.41
11	94520	1/2/2001	\$261,808.72
12	94490	12/31/2000	\$633,723.06
13	94460	1/2/2001	\$3,872.39
14	94430	1/2/2001	\$3,033.38
15	94410	1/2/2001	\$2,337.28

When you use Microsoft Internet Explorer and Excel 97, the Excel client opens in the background and the report launches in your browser. Depending on browser settings, you may see the Excel application open and minimized while viewing your report. Leave Excel open when viewing the spreadsheet. In Excel 97, you will be prompted to save the document as a Microsoft Excel Workbook with an .xls extension. This saves the file as a binary Excel document.

Reference: Limitations for FORMAT EXL97

- ❑ This format is compatible only with Excel 97 or higher. It is not compatible with Excel 95 or versions of Excel prior to Excel 95.
- ❑ While Excel 97 supports styling (StyleSheets), it does not support Cascading Style Sheets (CSS). Styling specified in a report that uses CSS (SET HTMLCSS=ON) will not be respected in Excel 97. The WRAP=*n* feature is not supported with EXL97, since this feature requires CSS.
- ❑ Numeric and date formatting options not supported for EXL2K are also not supported for EXL97. In addition, negative numbers displayed with brackets and trailing zeroes after the decimal are not supported. Note that dates are typically translated into Excel's General format, which may cause problems with sorting and other Excel features.
- ❑ PivotTables are not supported with EXL97. (EXL2K PIVOT is the only valid PivotTable format.)

Working With PostScript and PDF Reports**In this section:**

Creating Compound PDF or PostScript Reports

Adding PostScript Type 1 Fonts for PS and PDF Formats

Creating PDF Files for Use With UNIX Systems

Displaying An and AnV Fields With Line Breaks

Reference:

Required Files and DDNAMES

FOCUS generates a PDF or PS document from scratch. In order to do so, it must physically embed all the objects it displays or prints, including images and fonts, in the document itself.

When you execute a report request and specify PDF or PS as your format, FOCUS retrieves the data and begins to format the report. Fonts and images specified in the StyleSheet must be available to create the output file. FOCUS reads the font information from font files and embeds that information in the document.

To ensure that FOCUS can locate the required information, you must define and map it in the following files:

- ❑ **Font file, usually a PFB (Printer Font Binary) file.** This file contains the information about the shape to draw for each character of the font. The information in the font file is scalable, which means that a single font file can be used to generate characters of any size. Note, however, that bold and italic variations of the typeface are separate fonts. An alternative ASCII format, PFA, can also be used by FOCUS.

Note that the basic fonts delivered with FOCUS do not need PFA or PFB files. Acrobat Reader and PostScript printers already have the descriptions of these fonts.

- ❑ **Adobe Font Metrics (AFM) file.** This file is distributed with all Adobe fonts. It contains information about the size of each character in each font. FOCUS uses this information to lay out the report on the page. (Note that the three built-in fonts also have AFM files, which are distributed with FOCUS. However, these fonts don't require font files, since the fonts are built into Acrobat.)

Note: A Printer Metrics File (PFM) is also available. This file is used by applications such as Acrobat Reader for laying out text; however, it is not supported by FOCUS. You must use the AFM file.

- ❑ **FOCUS Font Map files.** These configuration files map the name of a font to the appropriate font and font metrics files (AFM and PFB or PFA). There are two versions of this file: **PDF** and **PSCRIPT**. You can update either one or both, depending on the output format (PDF or PS) with which you plan to use the Type 1 fonts.

Reference: Required Files and DDNAMES

When you produce a PostScript or PDF report, you need the following files:

Name	Purpose	z/OS	CMS
StyleSheet files. You can create them with a text editor (see Styling Reports).	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 and PDF output.	Member names start with PS. Allocated to ddname ERRORS.	File type AFM, any file name

Name	Purpose	z/OS	CMS
Font location file (supplied with FOCUS).	Maps font names to the Font Metrics files.	Member PSCRIPT (for PostScript) or PDF (for PDF) in the PDS allocated to ddname ERRORS.	File name PSCRIPT (for PostScript) or PDF (for PDF), File type FOCFTMAP
Output files. You create these with a HOLD, SAVE, or SET command.	Contain the formatted output.	DDNAME is HOLD or the AS name assigned in the HOLD command.	File type PS or PDF, any file name

Note that if you add fonts, you also need PFA or PFB files for those fonts.

Creating Compound PDF or PostScript Reports

How to:

Display Compound Reports

Compound reports combine multiple reports into a single PDF or PS file. This enables you to concatenate reports with styled formats (such as PDF, HTML, PS, or EXL2K). You can also embed image files in a compound report.

The first PDF or PS report defines the format for the concatenated report, enabling you to intersperse intermediate reports in other formats into one encompassing report. Using compound reports, you can gather data from different data sources and combine reports into one governing report that runs each request and concatenates the output into a single PDF or PS file.

Syntax: How to Display Compound Reports

For a compound report that may contain different report types, use the syntax

```
SET COMPOUND= {OPEN|CLOSE} [NOBREAK]
```

or

```
ON TABLE SET COMPOUND {OPEN|CLOSE}
```

Note that when you are using this syntax, you must also include the following code to identify the display format of each of the reports to be concatenated:

```
ON TABLE {HOLD|SAVE} [AS name] FORMAT formatname
```

If all of the reports in the compound set are of the same type, either PDF or PS, you can use the following, more compact, syntax

```
ON TABLE {HOLD|SAVE} [AS name] FORMAT {PDF|PS} {OPEN|CLOSE} [NOBREAK]
```

where:

name

Is the name of the generated file. The name is taken from the first request in the compound report. If no name is specified in the first report, the name HOLD is used.

OPEN

Is specified with the first report, and begins the concatenation process. A report that contains the OPEN attribute must be PDF or PS format.

CLOSE

Is specified with the last report, and ends the concatenation process.

NOBREAK

Is an optional phrase that suppresses page breaks. By default, each report is displayed on a separate page.

You can use NOBREAK selectively in a request to control which reports are displayed on the same page.

Note:

- ❑ Compound reports cannot be nested.
- ❑ You can save or hold the output from a compound report.

Example: Creating a Compound PDF Report

The following illustrates how to combine three separate PDF reports into one by creating a compound report. Notice that:

- ❑ Report 1 specifies ON TABLE HOLD FORMAT PDF OPEN. This defines the report as the first report and sets the format for the entire compound report as PDF.
- ❑ Report 2 specifies only the format ON TABLE HOLD FORMAT PDF.
- ❑ Report 3 specifies ON TABLE HOLD FORMAT PDF CLOSE. This defines the report as the last report.

These reports are all set to PDF format. When creating compound reports, however, only the first must be in either PDF or PS format; the rest can be in any styled format.

Report 1:

```

SET PAGE-NUM=OFF
TABLE FILE CENTORD
HEADING
"Sales Report"
" "
SUM LINEPRICE
BY PRODCAT
ON TABLE SET STYLE *
TYPE=HEADING, SIZE=18, $
ENDSTYLE
ON TABLE HOLD FORMAT PDF OPEN NOBREAK
END

```

Report 2:

```

TABLE FILE CENTORD
HEADING
"Inventory Report"
" "
SUM QUANTITY
BY PRODCAT
ON TABLE SET STYLE *
TYPE=HEADING, SIZE=18, $
ENDSTYLE
ON TABLE HOLD FORMAT PDF NOBREAK
END

```

Report 3:

```

TABLE FILE CENTORD
HEADING
"Cost of Goods Sold Report"
" "
SUM LINE_COGS
BY PRODCAT
ON TABLE SET STYLE *
TYPE=HEADING, SIZE=18, $
ENDSTYLE
ON TABLE HOLD FORMAT PDF CLOSE
END

```

The output appears as a PDF report. Because the syntax for reports 1 and 2 contain the NOBREAK command, all three reports appear on a single page. (Without NOBREAK, each report would appear on a separate page.)

Sales Report

<u>Product Category:</u>	<u>Line Total</u>
Cancorders	\$333,884,927.52
Cameras	\$9,718,752.55
CD Players	\$40,694,863.53
Digital Tape Recorders	\$38,773,229.69
DVD	\$53,182,674.89
PDA Devices	\$233,967,566.49
VCRs	\$23,801,009.34

Inventory Report

<u>Product Category:</u>	<u>Quantity:</u>
Cancorders	950,700
Cameras	98,141
CD Players	308,274
Digital Tape Recorders	524,027
DVD	241,287
PDA Devices	610,844
VCRs	170,977

Cost of Goods Sold Report

<u>Product Category:</u>	<u>Line Cost Of Goods Sold</u>
Cancorders	299,487,742.00
Cameras	8,944,891.00
CD Players	30,519,126.00
Digital Tape Recorders	36,157,863.00
DVD	47,275,593.00
PDA Devices	202,123,656.00
VCRs	22,056,033.00

Adding PostScript Type 1 Fonts for PS and PDF Formats

How to:

Configure Type 1 Postscript Fonts

Reference:

Support for the Symbol Font

You can add and configure PostScript Type 1 fonts to significantly expand your options for displaying and printing PS and PDF reports, beyond those provided by the basic set of fonts distributed with Adobe Acrobat Reader. Thousands of PostScript fonts are available to make your reports more stylish and useful, including some that support symbols and bar codes.

Through a simple process, you can customize your environment to take advantage of these fonts.

- ❑ First, copy the font files (AFM and PFB or PFA) to FOCUS.
- ❑ Next, update the FOCUS font map files (PDF and/or PSCRIPT). These configuration files already exist in the correct location, with specifications for the fonts that are distributed with Adobe Acrobat Reader. These files map the name of a font to its actual font file information. You will add the new font definitions to these files.

Once this step is complete, you can begin using the new font in your StyleSheet declarations.

You can also use a variety of utilities to convert Windows True Type fonts (such as Arial and Tahoma) into Type 1 fonts. Verify that you are licensed for this type of font use. Then, once converted, you can define and map these fonts for use by FOCUS.

One such utility is TTF2PT1.

For information about the Windows version, go to:

<http://gnuwin32.sourceforge.net/packages/ttf2pt1.htm>

Reference: Support for the Symbol Font

To use the Symbol font, specify font=symbol in your FOCUS StyleSheet:

- ❑ Some versions of Firefox 3 do not support the symbol font and substitute it with another font; for information about Firefox support for the symbol font, refer to Firefox sources.
- ❑ The Euro character displays in PDF output because the Adobe Symbol character set includes the Euro character.
- ❑ The Euro character does not display in DHTML and PPT report output because the Windows Symbol character set does not include the Euro character.
- ❑ The following style options can be rendered with the Symbol font:
 - ❑ DHTML and PPT support style=normal, bold, italic, and bold+italic.
 - ❑ PDF supports only style=normal. Any other style specified in the StyleSheet will be mapped to normal.

Procedure: How to Configure Type 1 Postscript Fonts

Once you have located the font files you wish to add, you can set up FOCUS to use one or more Type 1 fonts.

1. Copy the AFM file into the ERRORS data set on z/OS (or into a PDS with the same DCB attributes that's concatenated to it). On z/VM, copy the AFM file to a file with FILETYPE ERRORS.

Note that some of the comment lines may be truncated to 80 characters; this will not affect processing.

You can copy this file using FTP from Windows (in standard ASCII mode). The member name in the PDS or file name on VM should match the METRICSFILE name in the font map file.

2. You can use either PFB (binary) fonts or PFA (ASCII) fonts:
 - ❑ If you're using PFB (binary) fonts, create a partitioned data set on z/OS, put the PFB file in it (for example, using FTP in BINARY mode), and allocate this to DDNAME PFB. On VM, store it in a file with FILETYPE PFB.

The PDS should be created with the following DCB attributes:

```
RECFM: FB      LRECL: 1024      BLKSIZE: 27648
```

The member name on z/OS or the file name on z/VM should match the FONTFILE name in the font map file. For example, PREFIX.PFBFILES.DATA(OCRA).

- ❑ If you're using PFA (ASCII) font files on z/OS, create a PDS, put the PFA file in it (for example, using FTP in ASCII mode), and allocate this to DDNAME PFA. On VM, store it in a file with FILETYPE PFA. The PDS should be created with the following DCB attributes:

```
RECFM: VB      LRECL: 516       BLKSIZE: 27998
```

The member name in the PDS or the file name on VM should match the FONTFILE name in the font map file. For example, PREFIX.PFAFILES.DATA(OCRA).

Note that you can use PFB and PFA files simultaneously. The three character file type in the font map file (PFB or PFA) tells FOCUS which PDS to search for the specified member name.

3. Open the FOCUS font map file (on z/OS, members PDF or PSCRIPT in the PDS allocated to DDNAME ERRORS; on VM, file PSCRIPT ERRORS or PDF ERRORS) in a text editor.

Note: If the Windows font file names contain underscore characters, you must rename them, since underscore characters are not valid in z/OS member names or VM file names. For example, if the Windows designation is `font=garamond_light`, z/OS references would be as follows:

```
FONTFILENAME=garalt pdb *,  
METRICFILENAME= garalt afm*, $
```

4. Add a separate declaration for each font and style. A line in the FOCUS font map file may not exceed 80 characters. You can break the line after any comma.

The syntax is

```
font=fontname, style=style, metricsfile=METRICSFILENAME AFM *,
fontfile=FONTFILENAME PFB *, $
```

where:

fontname

Is the name of a Type 1 font. This name is used to reference the font in the FOCUS StyleSheet.

style

Is a font style. Normal is the default. This entry is used to reference the font style in the FOCUS StyleSheet.

If you wish to specify font style variations (for example, bold, italic, bold-italic for a particular font), you must purchase the font files for those styles and create a separate entry for each one in the FOCUS font map file.

METRICSFILENAME

Maps the font to the name of the font metrics file. The name of the metrics file must be followed by a space, followed by AFM, followed by an asterisk (*).

This entry must be in upper case and may not exceed 8 characters.

FONTFILENAME

Maps the font to the name of the font file. The name of the font file must be followed by a space, followed by PFB (or PFA), followed by asterisk (*).

The entry must be in upper case and may not exceed 8 characters.

Tip: If you are familiar with FOCUS StyleSheets, notice that the syntax of these files is similar. Lines can be continued after a comma and statements are terminated by a comma and dollar sign (,\$).

Example: FOCUS Font Map File Entries for PDF

Initially, this file contains mappings for the default fonts delivered with Adobe Acrobat Reader. These core Type 1 fonts are mapped to the appropriate font files. Each "style" of the font requires a separate font file, as shown for the core fonts. Notice that each one has four different sets of files for normal, bold, italic, and bold plus italic. You will need to follow the same model when you add a new Type 1 font to the file. If you want to add the normal and the bold versions of a new Type 1 font, you must make both sets of files available to FOCUS and map them in the font map file.

Tip: The file also contains a series of aliases (not shown) that map common Windows True Type fonts to existing Type 1 fonts (for example, Arial to Helvetica) along with a series of fonts used for various languages that FOCUS supports. You can set up your own aliases to ensure proper mapping and interpretation of Windows True Type Fonts.

Font metrics are already defined for the default fonts ,so there is not need to specify the PFB file name in the map file. However, you will need to specify that file, along with the AFM file, for each new font.

```
$ PDF.FMP: StyleSheets Font Map file for PDF Driver $
$ Version 1: All fonts mapped into Courier, Helvetica or Times.
$ "Native" Acrobat fonts:

font=Courier,    style=normal, metricsfile=PSCOUR AFM *, $
font=Courier,    style=bold,   metricsfile=PSCOURB AFM *, $
font=Courier,    style=italic, metricsfile=PSCOURI AFM *, $
font=Courier,    style=bold+italic, metricsfile=PSCOURBI AFM *, $

font=Helvetica, style=normal, metricsfile=PDHELV AFM *, $
font=Helvetica, style=bold,   metricsfile=PDHELVB AFM *, $
font=Helvetica, style=italic, metricsfile=PDHELVI AFM *, $
font=Helvetica, style=bold+italic, metricsfile=PDHELVBI AFM *, $

font=Times,      style=normal, metricsfile=PDTIME AFM *, $
font=Times,      style=bold,   metricsfile=PDTIMEB AFM *, $
font=Times,      style=italic, metricsfile=PDTIMEI AFM *, $
font=Times,      style=bold+italic, metricsfile=PDTIMEBI AFM *, $
```

-* Following is an entry for a new Type 1 Font: OCRA

A quick way to add an entry is to copy and modify an existing one.

```
font=OCRA, style=normal, metricsfile=OCRA AFM *, fontfile=OCRA PFB *, $
font=OCRA, style=bold, metricsfile=OCRAB AFM *, fontfile=OCRAB PFB *, $
```

FOCUS StyleSheet Declaration

The following is an in-line FOCUS StyleSheet declaration that formats the entire report to use the new Type1 font, OCRA. Other than the font, the report uses default styles:

```
ON TABLE SET STYLE *
TYPE=REPORT, FONT=OCRA, $
ENDSTYLE
```

Example: FOCUS Font Map File Entries for PS

The FOCUS font map file for PS includes a wide range of fonts that are typically supported by Level 2 Printers. To take advantage of a listed font, it must be installed on your printer.

These core Type 1 fonts are mapped to the appropriate font files. Each "style" of the font requires a separate font file, as shown for the core fonts. Notice that each one has four different sets of files for normal, bold, italic, and bold plus italic. You will need to follow the same model when you add a new Type 1 font to the file. If you want to add the normal and the bold versions of a new Type 1 font, you must make both sets of files available to FOCUS and map them in the font map file.

```
font=Arial,      style=normal,      metricsfile=PSHELV AFM *, $
font=Arial,      style=bold,        metricsfile=PSHELVB AFM *, $
font=Arial,      style=italic,      metricsfile=PSHELVI AFM *, $
font=Arial,      style=bold+italic, metricsfile=PSHELVBI AFM *, $

font=Avant Garde Gothic, style=normal,      metricsfile=PSAVAN AFM *, $
font=Avant Garde Gothic, style=bold,        metricsfile=PSAVANB AFM *, $
font=Avant Garde Gothic, style=italic,      metricsfile=PSAVANI AFM *, $
font=Avant Garde Gothic, style=bold+italic,
    metricsfile=PSAVANBI AFM *, $

font=Bookman,    style=normal,      metricsfile=PSBOOK AFM *, $
font=Bookman,    style=bold,        metricsfile=PSBOOKB AFM *, $
font=Bookman,    style=italic,      metricsfile=PSBOOKI AFM *, $
font=Bookman,    style=bold+italic, metricsfile=PSBOOKBI AFM *, $

font=Courier,    style=normal,      metricsfile=PSCOUR AFM *, $
font=Courier,    style=bold,        metricsfile=PSCOURB AFM *, $
font=Courier,    style=italic,      metricsfile=PSCOURI AFM *, $
font=Courier,    style=bold+italic, metricsfile=PSCOURBI AFM *, $

font=Courier New, style=normal,      metricsfile=PSCOUR AFM *, $
font=Courier New, style=bold,        metricsfile=PSCOURB AFM *, $
font=Courier New, style=italic,      metricsfile=PSCOURI AFM *, $
font=Courier New, style=bold+italic, metricsfile=PSCOURBI AFM *, $
```

```
font=Helvetica, style=normal, metricsfile=PSHELV AFM *, $
font=Helvetica, style=bold, metricsfile=PSHELVB AFM *, $
font=Helvetica, style=italic, metricsfile=PSHELVI AFM *, $
font=Helvetica, style=bold+italic, metricsfile=PSHELVBI AFM *, $

font=Helvetica Narrow, style=normal, metricsfile=PSNHLE AFM *, $
font=Helvetica Narrow, style=bold, metricsfile=PSNHLEB AFM *, $
font=Helvetica Narrow, style=italic, metricsfile=PSNHLEI AFM *, $
font=Helvetica Narrow, style=bold+italic, metricsfile=PSNHLEBI AFM *, $

font=Lubalin Graph, style=normal, metricsfile=PSLUB AFM *, $
font=Lubalin Graph, style=bold, metricsfile=PSLUBB AFM *, $
font=Lubalin Graph, style=italic, metricsfile=PSLUBI AFM *, $
font=Lubalin Graph, style=bold+italic, metricsfile=PSLUBBI AFM *, $

font=New Century Schoolbook, style=normal, metricsfile=PSSCHL AFM *, $
font=New Century Schoolbook, style=bold, metricsfile=PSSCHLB AFM *, $
font=New Century Schoolbook, style=italic, metricsfile=PSSCHLI AFM *, $
font=New Century Schoolbook, style=bold+italic,
metricsfile=PSSCHLBI AFM *, $

font=Palatino, style=normal, metricsfile=PSPALA AFM *, $
font=Palatino, style=bold, metricsfile=PSPALAB AFM *, $
font=Palatino, style=italic, metricsfile=PSPALAI AFM *, $
font=Palatino, style=bold+italic, metricsfile=PSPALABI AFM *, $

font=Souvenir, style=normal, metricsfile=PSSOUV AFM *, $
font=Souvenir, style=bold, metricsfile=PSSOUVB AFM *, $
font=Souvenir, style=italic, metricsfile=PSSOUVI AFM *, $
font=Souvenir, style=bold+italic, metricsfile=PSSOUVBI AFM *, $

font=Times, style=normal, metricsfile=PSTIME AFM *, $
font=Times, style=bold, metricsfile=PSTIMEB AFM *, $
font=Times, style=italic, metricsfile=PSTIMEI AFM *, $
font=Times, style=bold+italic, metricsfile=PSTIMEBI AFM *, $

font=Times New Roman, style=normal, metricsfile=PSTIME AFM *, $
font=Times New Roman, style=bold, metricsfile=PSTIMEB AFM *, $
font=Times New Roman, style=italic, metricsfile=PSTIMEI AFM *, $
font=Times New Roman, style=bold+italic, metricsfile=PSTIMEBI AFM *, $

font=Zapf Chancery, style=italic, metricsfile=PSZAPFCI AFM *, $
```

-* Following is an entry for a new Type 1 Font: OCRA

A quick way to add an entry is to copy and modify an existing one.

```
font=OCRA, style=normal, metricsfile=OCRA AFM *, fontfile=OCRA PFB *, $
font=OCRA, style=bold, metricsfile=OCRAB AFM *, fontfile=OCRAB PFB *, $
```

FOCUS StyleSheet Declaration

The following is an in-line FOCUS StyleSheet declaration that formats the entire report to use the new Type1 font, OCRA. Other than the font, the report uses default styles:

```
ON TABLE SET STYLE *
TYPE=REPORT, FONT=OCRA, $
ENDSTYLE
```

Creating PDF Files for Use With UNIX Systems

How to:

Specify Line Termination Characters When Creating a PDF File

Reference:

Required PDFLINETERM Settings Based on Environment

PDF files created with HOLD FORMAT PDF present a challenge if you work in an MVS or VM environment and use UNIX-based systems as the server for Adobe or as an intermediate transfer point.

The end of each PDF file has a table containing the byte offset, including line termination characters, of each PDF object in the file. The offsets indicate that each line is terminated by two characters, a carriage return and a line feed, which is the standard Windows text file format. However, records in a UNIX text file are terminated by one character, a line feed only. When using default settings, the offsets in a PDF file will be incorrect, causing an error when Acrobat attempts to open the file. If the file is then transferred in BINARY mode to Windows, it cannot be opened in Acrobat for Windows, as the carriage-return character was not inserted.

One solution has been to transfer the file to the UNIX system in text mode and then transfer in text mode to the Windows system, as the carriage return is added by the transfer facility when transferring to Windows.

If that is not possible or desirable, you can use the SET PDFLINETERM=SPACE command to facilitate binary transfer to Windows from an ASCII-based UNIX system. This command causes an extra space character to be appended to each record of the PDF output file. This extra space acts as a placeholder for the expected carriage return character and makes the object offsets in the file correct when it is transferred from MVS or VM to a UNIX system. This enables a UNIX server to open a PDF file in that environment.

Note: A text mode transfer is always required when transferring a text file from a mainframe to any other environment (Windows, ASCII Unix, or EBCDIC Unix).

Syntax: How to Specify Line Termination Characters When Creating a PDF File

In a profile, a FOCEXEC, or from the command line, issue the following command:

```
SET PDFLINETERM={STANDARD|SPACE}
```

In a TABLE request, issue the following command:

```
ON TABLE SET PDFLINETERM {STANDARD|SPACE}
```

where:

STANDARD

Creates a PDF file without any extra characters. This file will be a valid PDF file if transferred in text mode to a Windows machine, but not to a UNIX machine. If subsequently transferred from a UNIX machine to a Windows machine in text mode, it will be a valid PDF file on the Windows machine.

SPACE

Creates a PDF file with an extra space character appended to each record. This file will be a valid PDF file if transferred in text mode to a UNIX machine, but not to a Windows machine. If subsequently transferred from an ASCII UNIX machine to a Windows machine in binary mode, it will be a valid PDF file on the Windows machine.

Reference: Required PDFLINETERM Settings Based on Environment

The following chart will assist you in determining the correct setting to use, based on your environment:

Transferring from MVS or VM to:	SET PDFLINETERM=
EBCDIC UNIX (text transfer)	SPACE
ASCII UNIX (text transfer)	SPACE
ASCII UNIX (text); then to Windows (binary)	SPACE
UNIX (text); then to Windows (text)	STANDARD
Directly to Windows (text)	STANDARD

Displaying An and AnV Fields With Line Breaks

How to:

Display An and AnV Fields Containing Line Breaks on Multiple Lines

Using StyleSheet attributes, you can display An (character) and AnV (varchar) fields that contain line breaks on multiple lines in a PDF or PostScript report. Line breaks can be based on line feeds, carriage returns, or a combination of both. If you do not add these StyleSheet attributes, all line feed and carriage return formatting within these fields will be ignored.

Syntax: How to Display An and AnV Fields Containing Line Breaks on Multiple Lines

```
TYPE=REPORT,LINEBREAK=' type ', $
```

where:

`REPORT`

Is the required component for the LINEBREAK attribute.

`' type '`

Specifies that line breaks will be inserted in a report based on the following:

`LF` inserts a line break after each line-feed character found in all An and AnV fields.

`CR` inserts a line break after each carriage-return character found in all An and AnV fields.

`LF CR` inserts a line break after each combination of a line-feed character followed by a carriage-return character found in all An and AnV fields.

`CR LF` inserts a line break after each combination of a carriage-return character followed by a line-feed character found in all An and AnV fields.

Note: The report output must be formatted as PDF or PostScript.

Example: Displaying an Alphanumeric Field With Line Breaks in a PDF Report

The following request defines an alphanumeric named ANLB field with a semi-colon in the middle. The CTRAN function then replaces the semi-colon with a carriage return character and stores this string in a field named ANLBC. On the report output, this field displays on two lines:

```
DEFINE FILE EMPLOYEE
ANLB/A40 = 'THIS IS AN An FIELD;WITH A LINE BREAK.';
ANLBC/A40 = CTRAN(40, ANLB, 094, 013 , ANLBC);
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME ANLBC
WHERE LAST_NAME EQ 'BLACKWOOD'
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT,LINEBREAK='CR', $
ENDSTYLE
END
```

The output is:

LAST_NAME	ANLBC
BLACKWOOD	THIS IS AN An FIELD WITH A LINE BREAK.

Example: Using an Alphanumeric Field With a Line Break in a Subfoot

The following request defines an alphanumeric named ANLB field with a semi-colon in the middle. The CTRAN function then replaces the semi-colon with a carriage return character and stores this string in a field named ANLBC. In the subfoot, this field displays on two lines:

```
DEFINE FILE EMPLOYEE
ANLB/A40 = 'THIS IS AN An FIELD;WITH A LINE BREAK.';
ANLBC/A40 = CTRAN(40, ANLB, 094, 013 , ANLBC);
END
TABLE FILE EMPLOYEE
PRINT FIRST_NAME
BY LAST_NAME
WHERE LAST_NAME EQ 'BLACKWOOD'
ON LAST_NAME SUBFOOT
" "
" <ANLBC "
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT,LINEBREAK='CR', $
ENDSTYLE
END
```

The output is:

LAST_NAME
BLACKWOOD

THIS IS AN An FIELD
WITH A LINE BREAK.

FIRST_NAME
ROSEMARIE

14 | Advanced StyleSheet Features

Some advanced StyleSheet features include positioning, arranging, and aligning elements, adding images, grids and borders, and linking in a report. These features are described in this chapter.

For information about creating a StyleSheet, identifying and styling report components, and choosing a styled output format, see [Styling Reports](#) on page 491.

Topics:

- ❑ Positioning a Report Component
- ❑ Arranging Pages and Columns on a Page
- ❑ Wrapping and Justifying Report Components
- ❑ Aligning Heading and Footing Elements
- ❑ Adding Grids and Borders
- ❑ Adding an Image to a Report
- ❑ Linking in a Report
- ❑ Working With Mailing Labels and Multi-Pane Pages

Positioning a Report Component

How to:

- Specify the Starting Position of a Column
- Set a Starting Position for a Heading or Footing
- Set a Starting Position for a Heading or Footing Element
- Add Blank Space Around a Report Component

Reference:

Positioning Attributes

A StyleSheet enables you to specify an absolute or relative starting position for a column, heading, or footing, or element in a heading or footing. You can also add blank space around a report component.

For a PDF, PS, or HTML report, you can use the POSITION attribute in a StyleSheet to specify a starting position for a heading or footing, expressed as a unit measurement. For HTML, this capability requires an internal Cascading Style Sheet. For details on selecting an alignment method, see [Aligning Heading and Footing Elements](#) on page 742.

In addition, for a PDF or PS report, you can use the POSITION attribute to specify an absolute or relative starting position for an element within a heading or footing, or to align an item in a heading or footing with a report column. An absolute starting position is the distance from the left margin of the report. A relative starting position is the distance from the preceding object. For the first item on a heading line this is the left margin of the report.

In an HTML report, you can use related syntax and an internal Cascading StyleSheet to position an image in a heading or footing.

Reference: Positioning Attributes

Attribute	Description	Applies to
POSITION	<p>Sets absolute or relative starting position of a column.</p> <p>An absolute position is the distance from the left margin of the printed paper.</p> <p>A relative position is the distance from the default position. After the first column, the default position is the end of the preceding column.</p>	PDF PS
TOPGAP BOTTOMGAP	Adds blank space to the top or bottom of a report.	PDF PS
LEFTGAP RIGHTGAP	Adds blank space to the left or right of a report.	PDF PS

Syntax: How to Specify the Starting Position of a Column

This syntax applies to a PDF or PS report.

`TYPE=REPORT, COLUMN=identifier, POSITION={+|-}position, $`

where:

identifier

Selects a single column and collectively positions the column title, data, and totals if applicable. For valid values, see [Identifying an Entire Report, Column, or Row](#) on page 527.

+

Starts the column at the specified distance to the right of the default starting position.

By default, text items and alphanumeric fields are left-justified in a column, and numeric fields are right-justified in a column.

-

Starts the column at the specified distance to the left of the default starting position.

It is possible to create a report in which columns overlap. If this occurs, simply adjust the values.

position

Is the desired distance, in the unit of measurement specified with the UNITS attribute.

Syntax: **How to Set a Starting Position for a Heading or Footing**

Use the following syntax to specify a starting position for an entire heading or footing in relation to the left margin of a report.

`TYPE = headfoot, POSITION = position, $`

where:

headfoot

Is the type of heading or footing. Valid values are TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD, and SUBFOOT.

position

Is the desired distance from the left, expressed by the UNITS attribute (the default is INCHES).

Note: In an HTML report, this syntax must be used in conjunction with an internal Cascading Style Sheet.

Syntax: **How to Set a Starting Position for a Heading or Footing Element**

For a PDF or PS report, use the following syntax to specify a starting position for a heading or footing element in relation to the preceding item

`TYPE = headfoot, [subtype,] POSITION = {+|-}option, $`

where:

headfoot

Is the type of heading or footing. Valid values are TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD, and SUBFOOT.

subtype

Are additional attributes that identify the report component. These options can be used separately or in combination, depending upon the degree of specificity you need to fully identify an element. Valid values are:

LINE which identifies a line by its position in a heading or footing. Identifying individual lines enables you to format each line differently.

If a heading or footing has multiple lines and you apply a StyleSheet declaration that does not specify `LINE`, the declaration is applied to *all* lines. Blank lines are counted when interpreting the value of `LINE`.

`ITEM` which identifies an item by its position in a line. To divide a heading or footing line into items, you can use the `<+0>` spot marker.

To determine an `ITEM` number for an `OBJECT`, follow these guidelines:

- ❑ When used with `OBJECT=TEXT`, count only the text strings from left to right.
- ❑ When used with `OBJECT=FIELD`, count only values from left to right.
- ❑ When used without `OBJECT`, count text strings and field values from left to right.

If you apply a StyleSheet declaration that specifies `ITEM`, the number is counted from the beginning of each line in the heading or footing, not just from the beginning of the first line.

`OBJECT` which identifies an element in a heading or footing as a text string or field value. Valid values are `TEXT` or `FIELD`. `TEXT` may represent free text or a Dialogue Manager ampersand (&) variable.

It is not necessary to specify `OBJECT=TEXT` unless you are styling both text strings and embedded fields in the same heading or footing.

option

Is the alignment method. Valid values are:

`position` which is the desired distance, expressed by the `UNITS` attribute (the default is inches) for absolute positioning.

`+` which starts the heading or footing element at the specified distance to the *right* of the preceding item. For the first item in a heading or footing, the preceding item is the left margin of the report.

`-` which starts the heading or footing element at the specified distance to the *left* of the preceding item. This is useful if you want to overlap images in a heading.

`column_title` which aligns the heading or footing element with the first character of the designated column.

Example: Specifying an Absolute Starting Position for a Column

The following illustrates how to position a column in a printed report. The request specifies that the PRODUCT_DESCRIPTION field should appear three inches from the left margin of the PDF report.

```
TABLE FILE GGORDER
"PRODUCTS ORDERED ON 08/01/96"
SUM QUANTITY BY PRODUCT_DESCRIPTION
WHERE ORDER_DATE EQ '080196'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT, COLUMN=PRODUCT_DESCRIPTION, POSITION=3, $
ENDSTYLE
END
```

The output is:

```
PRODUCTS ORDERED ON 08/01/96
```

<u>Product</u>	<u>Ordered Units</u>
Biscotti	6140
Coffee Grinder	2493
Coffee Pot	3100
Croissant	7465
French Roast	12965
Hazelnut	4186
Kona	2591
Mug	5332
Scone	5949
Thermos	2362

Example: Specifying a Relative Starting Position for a Column

This request positions the column title and data for the QUANTITY field two inches from the default position; in this case, two inches from the end of the preceding column.

```
TABLE FILE GGORDER
"PRODUCTS ORDERED ON 08/01/96"
SUM QUANTITY BY PRODUCT_DESCRIPTION
WHERE ORDER_DATE EQ '080196'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT, COLUMN=PRODUCT_DESCRIPTION, POSITION=3, $
TYPE=REPORT, COLUMN=QUANTITY, POSITION=+2, $
ENDSTYLE
END
```

QUANTITY, titled Ordered Units in the report, is relatively positioned to Product:

```
PRODUCTS ORDERED ON 08/01/96
```

<u>Product</u>	<u>Ordered Units</u>
Biscotti	6140
Coffee Grinder	2493
Coffee Pot	3100
Croissant	7465
French Roast	12965
Hazelnut	4186
Kona	2591
Mug	5332
Scone	5949
Thermos	2362

Example: Setting an Absolute Starting Position for a Heading Item

This request uses the spot marker <+0> to divide the report heading into three text strings. It starts the third text string, 1st Qtr 2001, 3 inches from the left report margin. This technique can be used in PDF as well as PS reports.

```
TABLE FILE GGSALES
SUM UNITS DOLLARS BY CATEGORY BY PRODUCT
ON TABLE SUBHEAD
"Sales Report - <+0>All Products<+0> 1st Qtr 2001"
" "
ON TABLE SET PAGE-NUM OFF
ON TABLE SET SQUEEZE ON
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLESHEET *
TYPE = TABHEADING, OBJECT = TEXT, ITEM=1, SIZE = 12, STYLE = BOLD, $
TYPE = TABHEADING, OBJECT = TEXT, ITEM=2, STYLE = BOLD, $
TYPE = TABHEADING, OBJECT = TEXT, ITEM=3, POSITION = 3, $
ENDSTYLE
END
```

The output is:

Sales Report - All Products		1st Qtr 2001	
<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
Coffee	Capuccino	189217	2381590
	Espresso	308986	3906243
	Latte	878063	10943622
Food	Biscotti	421377	5263317
	Croissant	630054	7749902
	Scone	333414	4216114
Gifts	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829

Example: Setting a Relative Starting Position for a Heading Item

This request uses the spot marker <+0> to divide the report heading into three text strings. It starts the third text string, 1st Qtr 2001, one inch to the right of the previous item on the heading line. Inches are the default unit of measure. This technique can be used in PDF as well as PS reports.

```
TABLE FILE GGSales
SUM UNITS DOLLARS BY CATEGORY BY PRODUCT
ON TABLE SUBHEAD
"Sales Report - <+0>All Products<+0> 1st Qtr 2001"
" "
ON TABLE SET PAGE-NUM OFF
ON TABLE SET SQUEEZE ON
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLESHEET *
TYPE = TABHEADING, OBJECT = TEXT, ITEM=1, SIZE = 12, STYLE = BOLD, $
TYPE = TABHEADING, OBJECT = TEXT, ITEM=2, STYLE = BOLD, $
TYPE = TABHEADING, OBJECT = TEXT, ITEM=3, POSITION = +1, $
ENDSTYLE
END
```

The output is:

Sales Report - All Products		1st Qtr 2001	
<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
Coffee	Capuccino	189217	2381600
	Espresso	308986	3906243
	Latte	878063	10943622
Food	Biscotti	421377	5263328
	Croissant	630054	7749902
	Scone	333414	4216114
Gifts	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829

Example: Aligning a Heading Item With a Column

This request uses the spot marker <+0> to divide the report heading into three text strings. It starts the second text string at the horizontal position where the column UNITS (Unit Sales) is. This technique can be used in PDF as well as PS reports.

```
TABLE FILE GGSales
SUM UNITS DOLLARS BY CATEGORY BY PRODUCT
ON TABLE SUBHEAD
"Sales Report - <+0>All Products<+0> 1st Qtr 2001"
" "
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PDF
ON TABLE SET SQUEEZE ON
ON TABLE SET STYLESHEET *
TYPE = TABHEADING, LINE=1, ITEM=2, POSITION=UNITS, $
ENDSTYLE
END
```

The output is:

Sales Report -		All Products 1st Qtr 2001	
<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
Coffee	Capuccino	189217	2381590
	Espresso	308986	3906243
	Latte	878063	10943622
Food	Biscotti	421377	5263317
	Croissant	630054	7749902
	Scone	333414	4216114
Gifts	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829

You can employ several types of spot markers to refine the positioning of headings and footings, and elements within them, in HTML and PDF reports that use proportional fonts. For maximum control, you can combine spot markers with other alignment techniques.

You can also use spot markers to position heading and footing elements at fixed and relative column locations. Several spot markers control positioning based on the pre-defined width of a character in a monospace font. This formatting technique is not supported for proportional fonts.

Syntax: How to Add Blank Space Around a Report Component

In a PDF or PostScript report, you can add space around report components.

Use the TOPGAP and BOTTOMGAP attributes to control spacing between heading or footing lines, and between heading or footing text and the grid lines above and below them.

Note: You can use TOPGAP and BOTTOMGAP with multi-line headings. Keep in mind that between heading lines, the top *and* bottom gap will be inserted, making the spacing between lines greater than the spacing at the top and bottom of the heading.

```
TYPE=headfoot, {TOPGAP|BOTTOMGAP}=gap, $
TYPE=REPORT, {TOPGAP|BOTTOMGAP}=gap, $
  TYPE=type, [COLUMN=identifier,|ACROSSCOLUMN=acrosscolumn,]
  {LEFTGAP|RIGHTGAP}=gap, $
```

where:

TOPGAP

Indicates how much space to add above the report.

BOTTOMGAP

Indicates how much space to add below the report.

gap

Is the amount of blank space, in the unit of measurement specified with the UNITS attribute.

In the absence of grids or background color, the default value is 0. For RIGHTGAP, the default value is proportional to the size of the text font.

In the presence of grids or background color, the default value increases to provide space between the grid and the text or to extend the color beyond the text.

The gaps must be the same within a single column or row. That is, you cannot specify different left or right gaps for individual cells in the same column, or different top and bottom gaps for individual cells in the same row.

type

Identifies the report component. For valid values, see [Identifying Report Components](#) on page 525.

identifier

Selects one or more columns using the COLUMN attribute described in [Identifying an Entire Report, Column, or Row](#) on page 527.

acrosscolumn

Selects the same column under each occurrence of an ACROSS sort field, using the ACROSSCOLUMN attribute described in [Identifying an Entire Report, Column, or Row](#) on page 527.

LEFTGAP

Indicates how much space to add to the left of a report component.

RIGHTGAP

Indicates how much space to add to the right of a report component.

Example: Adding Blank Space Above Data Values

This request generates one-tenth of an inch of blank space above every data value in a PDF report.

```
SET PAGE-NUM = OFF
TABLE FILE GGORDER
"PRODUCTS ORDERED ON 08/01/96"
" "
SUM QUANTITY BY PRODUCT_DESCRIPTION
WHERE ORDER_DATE EQ '080196'
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=DATA, TOPGAP = 0.1, $
ENDSTYLE
END
```

The data is spaced for readability:

PRODUCTS ORDERED ON 08/01/96

<u>Product</u>	<u>Ordered Units</u>
Biscotti	6140
Coffee Grinder	2493
Coffee Pot	3100
Croissant	7465
French Roast	12965
Hazelnut	4186
Kona	2591
Mug	5332
Scone	5949
Thermos	2362

Example: Adding Blank Space to Separate Heading Text From Grid Lines in a PDF Report

This request generates a PDF report with blank space added above and below the report heading to separate the text from the upper and lower grid lines. The space above is added by the TOPGAP attribute; the space below is added by the BOTTOMGAP attribute.

```
TABLE FILE GGSales
SUM BUDUNITS UNITS BUDDOLLARS DOLLARS
BY CATEGORY
ON TABLE SUBHEAD
"SALES REPORT <+0>December 2001"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PDF
ON TABLE SET SQUEEZE ON
ON TABLE SET STYLESHEET *
TYPE = TABHEADING, GRID=ON, JUSTIFY=Center, TOPGAP=.25, BOTTOMGAP=.25, $
TYPE = TABHEADING, FONT='TIMES', SIZE=12, STYLE=BOLD, $
TYPE = TABHEADING, ITEM=2, SIZE=10, STYLE=ITALIC, $
ENDSTYLE
END
```

The output is:

SALES REPORT <i>December 2001</i>				
Category	Budget Units	Unit Sales	Budget Dollars	Dollar Sales
Coffee	1385923	1376266	17293886	17231455
Food	1377564	1384845	17267160	17229333
Gifts	931007	927880	11659732	11695502

Arranging Pages and Columns on a Page**In this section:**

Determining Column Width

Reference:

Column Arrangement Features

How easily a user locates data depends on the arrangement of columns on a page. You have many design options. Using StyleSheet attributes or commands, you can:

- Determine column width.

- ❑ Control the number of spaces between columns.
- ❑ Change the order of vertical sort (BY) columns.
- ❑ Stack columns to reduce report width, or to easily compare values in a report by creating a matrix.
- ❑ Specify the absolute or relative starting position for a column.

Reference: Column Arrangement Features

Feature	Description	Applies to
<code>SQUEEZE</code>	Sets column width.	HTML (requires FONT=DEFAULT-FIXED) PDF PS
<code>SET SPACES</code>	Sets number of spaces between columns.	HTML (requires SET STYLEMODE=FIXED)
<code>SEQUENCE</code>	Sets column order.	PDF PS
<code>FOLD-LINE</code>	Reduces report width by stacking columns.	PDF PS
<code>OVER</code>	Stacks columns by placing them over one another.	HTML PDF PS
<code>IN {n +n}</code>	Sets absolute or relative starting position of a column.	HTML (requires SET STYLEMODE=FIXED) PDF PS

For more information, see [Customizing Tabular Reports](#) on page 357.

Determining Column Width

How to:

Determine Column Width (HTML)

Determine Column Width (PDF or PS)

The value of the SQUEEZE attribute in a StyleSheet determines column width in a report. You can use a SET parameter instead of a StyleSheet to set the value of SQUEEZE. If there are conflicting StyleSheet and SET values, the StyleSheet overrides the SET.

When SQUEEZE is set to ON (the default), StyleSheet column width is ignored. Column width is determined using your browser's default settings for HTML. For PDF, column width is based on the widest data value or column title, whichever is greater.

SQUEEZE may affect the way headings, footings, and column titles display in your report.

Syntax: **How to Determine Column Width (HTML)**

This syntax applies to an HTML report. For the syntax for a PDF or PS report, see [How to Determine Column Width \(PDF or PS\)](#) on page 720.

```
[TYPE=REPORT, ] SQUEEZE={ON|OFF}, $
```

where:

TYPE=REPORT

Applies the column width to the entire report. Not required, as it is the default.

ON

Determines column width based on the widest data value or column title, whichever is greater. ON is the default value.

For HTML reports, the Web browser shrinks the column width to the shortest column title or field value.

OFF

Determines column width based on the field format in the Master File. Blank spaces pad the column width up to the length of the column title or field format, whichever is greater.

Example: Using Default Column Width (HTML)

This request uses SQUEEZE=ON (the default) for an HTML report. Column width is based on the wider of the data value or column title.

```
SET PAGE-NUM = OFF
TABLE FILE GGSALES
SUM UNITS DOLLARS
BY CATEGORY BY PRODUCT
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, FONT=COURIER, $
ENDSTYLE
END
```

For Category, Unit Sales, and Dollar Sales, the column title is wider than the corresponding data values. For Product, the wider data values determine column width. The HTML report is:

<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
Coffee	Capuccino	189217	2381590
	Espresso	308986	3906243
	Latte	878063	10943622
Food	Biscotti	421377	5263317
	Croissant	630054	7749902
	Scone	333414	4216114
Gifts	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829

Syntax: How to Determine Column Width (PDF or PS)

This syntax applies to a PDF or PS report. For the syntax for an HTML report, see [How to Determine Column Width \(HTML\)](#) on page 719.

```
[TYPE=REPORT,] COLUMN=identifier, SQUEEZE={ON|OFF|width}, $
```

where:

```
TYPE=REPORT
```

Applies the column width to the entire report. Not required, as it is the default.

identifier

Selects a column. If you omit a column identifier, the value for SQUEEZE applies to all columns in a report. You can also use SET SQUEEZE to set the width of all columns.

ON

Determines column width based on the widest data value or column title, whichever is greater. ON is the default value.

OFF

Determines column width based on the field format in the Master File. Blank spaces pad the column width up to the length of the column title or field format, whichever is greater. OFF is the default value.

width

Is a measurement for the column width, specified with the UNITS attribute.

If the widest data value exceeds the specified measurement:

And the field is ...	The following appears ...
Alphanumeric	As much of the value as will fit in the specified width, followed by an exclamation mark (!) to indicate truncation.
Numeric	Asterisks (*) in place of the field value.

Example: Determining Column Width (PDF)

This request uses SQUEEZE=2.5 to increase the default column width of the PRODUCT field in a PDF report. Note that this feature is used primarily for printed reports. Depending on your screen resolution, the column width may appear differently than it prints.

```
TABLE FILE GGSales
SUM UNITS
BY PRODUCT
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT, COLUMN=PRODUCT, SQUEEZE=2.5, $
ENDSTYLE
END
```

The PDF report is:

<u>Product</u>	<u>Unit Sales</u>
Biscotti	421377
Capuccino	189217
Coffee Grinder	186534
Coffee Pot	190695
Croissant	630054
Espresso	308986
Latte	878063
Mug	360570
Scone	333414
Thermos	190081

Wrapping and Justifying Report Components

In this section:

Controlling Wrapping of Report Data

Controlling Wrapping With Alternative Methods

Justifying Report Columns

Justifying a Heading or Footing

Justifying a Column Title

Justifying a Label for a Subtotal or Grand Total

You can position data within a report by specifying whether or not you wish to have data wrap within a cell or by selecting a justification (right, left or center) of a column or heading.

Controlling Wrapping of Report Data

How to:

Control Wrapping of Report Data Using a StyleSheet Attribute

Control Wrapping of Report Data Using a SET Command

Control Spacing Between Wrapped Lines

You can control the wrapping of report data in a report, thus preventing line breaks within report cells. When using HTML output, most Web browsers will, by default, wrap alphanumeric report data that does not fit on a single line in a cell.

This bumps the contents of the cell onto a second line. A Web browser wraps data based on its algorithmic settings. Use the WRAP attribute if you wish to suppress a Web browser's data wrapping.

For information about wrapping in Excel 2000 reports, see [Controlling Column Width and Wrapping](#) on page 644.

Syntax: **How to Control Wrapping of Report Data Using a StyleSheet Attribute**

To control wrapping of text inside a report, use the following syntax within a StyleSheet.

```
TYPE=type, [subtype,] WRAP=value, $
```

where:

type

Is the report component you wish to affect, such as REPORT, HEADING, or TITLE.

subtype

Is any additional attribute, such as COLUMN, ACROSS, ITEM, and so on, that is needed to identify the report component that you are formatting. See [Identifying Report Components](#) on page 525 for more information about how to specify different report components.

value

Is one of the following:

[ON](#) turns on data wrapping. This is the default.

[OFF](#) turns off data wrapping.

n represents a specific numeric value to which the column width can be set. The value represents the measure specified with the UNITS parameter.

Syntax: How to Control Wrapping of Report Data Using a SET Command

To control wrapping of text inside a report, use the following syntax

```
SET WRAP=value, $
```

where:

value

Is one of the following:

ON turns on data wrapping. This is the default.

OFF turns off data wrapping.

n represents a specific numeric value that the column width can be set to. The value represents the measure specified with the UNITS parameter.

Example: Allowing the Web Browser to Wrap Report Data

The following example, with WRAP=ON, wraps report data based on the Web browser's functionality. Note that because this value is the default, there is no need to specify WRAP=ON in the report request syntax.

```
TABLE FILE GGPRODS
PRINT SIZE UNIT_PRICE PACKAGE_TYPE
VENDOR_CODE VENDOR_NAME
BY PRODUCT_ID BY PRODUCT_DESCRIPTION
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
ENDSTYLE
END
```

The output is:

PAGE 1						
Product Code	Product	Size	Unit Price	Package	Vendor ID	Vendor Name
B141	Hazelnut	16	58.00	Pounds	V082	Coffee Connection
B142	French Roast	12	81.00	Pounds	V083	European Specialities,
B144	Kona	12	76.00	Pounds	V081	Evelina Imports, Ltd

Notice that records in the Vendor Name column break to a second line.

Syntax: How to Control Spacing Between Wrapped Lines

You can use the WRAPGAP attribute in a StyleSheet to control spacing between wrapped lines in PDF and PostScript report output.

`type=component , WRAPGAP={ON|OFF|n}`

where:

component

Is the component with wrapped lines.

ON

Does not leave any space between wrapped lines. ON is equivalent to specifying 0.0 for *n*.

OFF

Places wrapped data on the next line. OFF is the default value.

n

Is a number greater than or equal to zero that specifies how much space to leave between wrapped lines (using the unit of measurement specified by the UNITS attribute). Setting *n* to zero does not leave any space between wrapped lines, and is equivalent to specifying WRAPGAP=ON.

Example: Specifying Spacing for Wrapped Lines

In the following request, wrapping is turned on for the ADDRESS_LN3 column of the report:

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN3
BY LAST_NAME BY FIRST_NAME
WHERE LAST_NAME LE 'CROSS'
  ON TABLE HOLD FORMAT PDF
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
type=report, grid=on, $
type=data, topgap=0.2, bottomgap=0.2, $
type=data, wrapgap=off, $
type=REPORT, column=ADDRESS_LN3, wrap=1.0 , $
END
```

With WRAPGAP=OFF, each wrapped line is placed on the next report line:

LAST NAME	FIRST NAME	ADDRESS LI
BANNING	JOHN	FREEPORT NY 11520
BLACKWOOD	ROSEMARIE	NEW YORK NY 10001
		EDISON NJ 08817
		BROOKLYN NY 11210
CROSS	BARBARA	NEW YORK NY 10001
		FLUSHING NY 11354

With WRAPGAP=ON, the wrapped lines are placed directly under each other:

LAST NAME	FIRST NAME	ADDRESS L!
BANNING	JOHN	FREEMPORT NY 11520
BLACKWOOD	ROSEMARIE	NEW YORK NY 10001
		EDISON NJ 08817
		BROOKLYN NY 11210
CROSS	BARBARA	NEW YORK NY 10001
		FLUSHING NY 11354

Reference: Usage Notes for WRAPGAP

- ❑ You can only specify WRAPGAP for columns that have wrapping enabled (WRAP attribute or parameter set to ON or a number). The TOPGAP and BOTTOMGAP attributes specify how much vertical space to leave above and below a report component. Increasing the values of these attributes makes a decrease in spacing between wrapped lines more noticeable.

Controlling Wrapping With Alternative Methods

How to:

Control Wrapping Using SQUEEZE

Control Wrapping Using SQUEEZE as a SET Command

Reference:

Set Commands that Affect Wrapping in a Report

Usage Notes for WRAP and SQUEEZE

Working With Multi-Table HTML Reports

In addition to the WRAP parameter, whether or not text wraps within a cell is affected by the SQUEEZE and GRID attributes. Both SQUEEZE and GRID can be used as SET commands or StyleSheet attributes. By specifying a setting other than the default value for these two attributes, the wrapping behavior in a browser is affected.

Reference: [Set Commands that Affect Wrapping in a Report](#)

The following SET commands affect the wrapping of data.

Command	Action
<code>SET SQUEEZE=OFF</code>	Forces the browser to display the entire column width and suppresses the wrapping of column data.
<code>SET SQUEEZE=ON</code>	Allows the browser to follow its own display behavior. If a browser's settings so specify, columns will be compressed to the length of the longest data value, and column data with embedded blanks will be wrapped.
<code>SET GRID=OFF</code>	Automatically suppresses the wrapping of data.
<code>SET GRID=ON</code>	Has no effect on either wrapping or compressing column width.

Syntax: How to Control Wrapping Using SQUEEZE

SQUEEZE can be used as either a SET parameter or a StyleSheet attribute. To control wrapping using SQUEEZE, use the following syntax within a StyleSheet.

```
TYPE =type, SQUEEZE=value, $
```

where:

type

Is the report component you wish to affect, such as REPORT, HEADING, or TITLE.

value

Is one of the following:

ON turns on data wrapping. This is the default for HTML output.

OFF turns off data wrapping. This is the default for PDF and PostScript.

n represents a specific numeric value to which the column width can be set (valid only in PDS and PostScript) The value represents the measure specified with the UNITS parameter.

Syntax: How to Control Wrapping Using SQUEEZE as a SET Command

To suppress the wrapping feature in a report by setting the SQUEEZE parameter to OFF, use the following syntax

```
SET SQUEEZE=value, $
```

where:

value

Is one of the following:

ON turns on data wrapping.

OFF turns off data wrapping. This is the default.

n represents a specific numeric value to which the column width can be set (valid only in PDS and PS). The value represents the measure specified with the UNITS parameter.

Note: SQUEEZE turns off data wrapping. If a data value is wider than the specified width of the column, it will be hidden from view. Column width can be adjusted in Excel after the spreadsheet has been generated.

Reference: Usage Notes for WRAP and SQUEEZE

Most Web browsers-by default-determine the width of a report column using an internal algorithm. If a data value exceeds the column's width, the browser wraps that data onto the next line of the column. If you want to prevent these line breaks, you can override this browser behavior by using the WRAP StyleSheet attribute.

Specifying a width is supported in HTML reports that generate an internal Cascading Style Sheet, as well as in PDF and PostScript reports.

If you do not specify the WRAP attribute in:

- ❑ An HTML report, it defaults to ON.
- ❑ A PDF report, it defaults to OFF.

Note that the WRAP and SQUEEZE attributes are incompatible: you should not apply both of them to the same report component.

The WRAP attribute is for alphanumeric columns, and is not supported for text (TX) columns.

Reference: Working With Multi-Table HTML Reports

- ❑ You can control where a report breaks using SET LINES or PAGE-BREAK in the request.
- ❑ If SET LINES=999999, the entire report is matched to a single displayed Web page, even if SET STYLEMODE=PAGED.
- ❑ ON *sortfield* PAGE-BREAK or BY *sortfield* PAGE-BREAK overrides a SET LINES command and breaks a report into multiple HTML tables whenever the sort field value changes.
- ❑ Column titles are generated for every PAGE-BREAK or according to the SET LINES parameter.
- ❑ When a report is broken into multiple HTML tables, the browser displays each table according to its own algorithm. Set SQUEEZE to OFF and/or WRAP to OFF to ensure that HTML tables are aligned consistently across pages.

Example: Displaying a Multiple-Table HTML Report

In this request, each page is returned to the browser as a separate HTML table. SQUEEZE is set to OFF for consistent alignment of tables across pages.

```

SET STYLEMODE = PAGED
SET LINES = 12
TABLE FILE CENTORD
HEADING
"SALES OVER $200,000"
PRINT LINEPRICE AS 'Sales'
BY SNAME BY ORDER_NUM
WHERE LINEPRICE GT 200000
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, SQUEEZE=OFF, $
ENDSTYLE
END

```

Two pages of the report follow, showing consistent alignment:

PAGE 19

SALES OVER \$200,000

<u>Store Name:</u>	<u>Order Number:</u>	<u>Sales</u>
Consumer Merchandise	57460	\$249,589.40
	73742	\$204,073.28
	74003	\$219,898.15
	77390	\$276,878.50
TV City	28605	\$210,990.34
	28696	\$203,493.89

PAGE 20

SALES OVER \$200,000

<u>Store Name:</u>	<u>Order Number:</u>	<u>Sales</u>
TV City	28870	\$237,226.33
	28891	\$202,903.10
	36819	\$260,287.49
		\$266,800.94
	37891	\$202,812.61
	38870	\$215,947.68

The same two pages illustrate inconsistent alignment with SQUEEZE set to ON:

PAGE 19

SALES OVER \$200,000

<u>Store Name:</u>	<u>Order Number:</u>	<u>Sales</u>
Consumer Merchandise	57460	\$249,589.40
	73742	\$204,073.28
	74003	\$219,898.15
	77390	\$276,878.50
TV City	28605	\$210,990.34
	28696	\$203,493.89

PAGE 20

SALES OVER \$200,000

<u>Store Name:</u>	<u>Order Number:</u>	<u>Sales</u>
TV City	28870	\$237,226.33
	28891	\$202,903.10
	36819	\$260,287.49
		\$266,800.94
	37891	\$202,812.61
	38870	\$215,947.68

Justifying Report Columns

How to:

Justify a Report Column

You can adjust text within a column by specifying whether report columns are left justified, right justified, or centered. By default, alphanumeric columns are left justified, numeric columns are right justified, and heading and footing elements are left justified. However, you can change the default using the JUSTIFY attribute. You can also justify column titles using the /R /L and /C formatting options in the report request.

Syntax: How to Justify a Report Column

To left justify, right justify, or center a column, use the following syntax within a StyleSheet.

```
TYPE=type, [subtype,] [COLUMN=column,] JUSTIFY=option, $
```

where:

type

Is the report component you wish to affect, such as REPORT, HEADING, or TITLE.

subtype

Is any additional attribute, such as COLUMN, ACROSS, ITEM, and so on, that is needed to identify the report component that you are formatting.

column

Is the column or group of columns you wish to justify. This attribute is only necessary if you wish to justify a specific column or set of columns. Omitting this attribute justifies the entire report.

option

Is the justification you wish to select:

`LEFT` specifies that the column will be left justified.

`RIGHT` specifies that the column will be right justified.

`CENTER` specifies that the column will be centered.

Example: Justifying Data in a Report Column

The following example displays the StyleSheet syntax used to center the data in the Vendor Name column. The header is also center justified.

```
TABLE FILE GGPRODS
HEADING
"PRODUCT REPORT"
SUM UNITS BY PRODUCT_DESCRIPTION BY PRODUCT_ID BY VENDOR_NAME
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, COLUMN=VENDOR_NAME, JUSTIFY=CENTER, $
TYPE=HEADING, JUSTIFY=CENTER, $
TYPE=REPORT, GRID=OFF, $
ENDSTYLE
END
```

The output is:

PRODUCT REPORT			
<u>Product</u>	<u>Product Code</u>	<u>Vendor Name</u>	<u>Unit Price</u>
Biscotti	F102	Delancey Bakeries	17.00
Coffee Grinder	G110	Appliance Craft	125.00
Coffee Pot	G121	Appliance Craft	140.00
Croissant	F103	West Side Bakers	28.00
French Roast	B142	European Specialities,	81.00
Hazelnut	B141	Coffee Connection	58.00
Kona	B144	Evelina Imports, Ltd	76.00
Mug	G100	NY Ceramic Supply	26.00
Scone	F101	Ridgewood Bakeries	13.00
Thermos	G104	ThermoTech, Inc	96.00

Justifying a Heading or Footing

How to:

Justify a Heading or Footing in a StyleSheet

Reference:

Justification Regions and Behavior

You can left justify, right justify, or center a heading or footing in a StyleSheet. By default, a heading or footing is left justified. In addition, you can justify an individual line or lines in a multiple-line heading or footing.

To center a page heading or footing over the report data, you can use a legacy formatting technique that does not require a StyleSheet; simply include the CENTER option in a HEADING or FOOTING command.

Justification behavior in HTML and PDF. For HTML reports, justification is implemented with respect to the report's width. That means a centered heading is centered over the report content. In contrast, for PDF reports, the default justification area is the page width, rather than the report width. This results in headings and footings that are not centered on the report. In most cases, you can achieve justification based on report width in a PDF report by adding the command `SET SQUEEZE=ON` to your request. This command improves the appearance of the report by eliminating excessive white space between columns and implements justification over the report content. However, if the heading is wider than the report, it will be centered on the page, even when `SQUEEZE=ON`.

Tip: You can also use justification syntax in combination with other StyleSheet syntax to align headings, footings, and items in them with other report elements, based on either unit measurements or relationships to other columns. For a summary of these options, see [Aligning Heading and Footing Elements](#) on page 742.

Syntax: **How to Justify a Heading or Footing in a StyleSheet**

```
TYPE = headfoot, [LINE = line_#,] JUSTIFY = option, $
```

where:

headfoot

Is the type of heading or footing. Valid values are TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD, and SUBFOOT.

line_#

Optionally identifies a line by its position in the heading or footing so that you can individually align it. If a heading or footing has multiple lines and you omit this option, the value supplied for JUSTIFY applies to all lines.

option

Is the type of justification. Valid values are:

`LEFT` which left justifies the heading or footing. This value is the default.

`RIGHT` which right justifies the heading or footing.

`CENTER` which centers the heading or footing.

Example: Centering All Lines in a Multiple-Line Report Heading

This request centers all lines in a multiple-line report heading using the single StyleSheet attribute for the entire heading:

```
TABLE FILE GGSales
SUM BUdunits units BUdDollars Dollars
BY Category
ON TABLE SUBHEAD
"SALES REPORT"
"**(CONFIDENTIAL)**"
"December 2001"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE = REPORT, GRID=OFF, $
TYPE = TABHEADING, JUSTIFY = CENTER, $
ENDSTYLE
END
```

The output is:

SALES REPORT				
(CONFIDENTIAL)				
December 2001				
<u>Category</u>	<u>Budget Units</u>	<u>Unit Sales</u>	<u>Budget Dollars</u>	<u>Dollar Sales</u>
Coffee	1385923	1376266	17293886	17231455
Food	1377564	1384845	17267160	17229333
Gifts	931007	927880	11659732	11695502

Tip: To run this report in PDF format, add the code ON TABLE SET SQUEEZE ON to eliminate excessive white space between columns and to center the heading over the report.

Reference: Justification Regions and Behavior

The region in which text is justified depends on the relationship of the sizes of certain elements in the report:

- When SQUEEZE=ON, the maximum width of all the heading types in the report is calculated. This value is called MaxHeadWidth.

If MaxHeadWidth is less than or equal to the total width of the columns of the report, headings are justified in the space over the report columns.

If MaxHeadWidth exceeds the total width of the columns of the report, headings are centered and right-justified in the entire width of the page.

- ❑ When SQUEEZE=OFF, the maximum width of all the headings are not precalculated; headings are centered in the entire width of the page.

With a styled, multiple-panel report (in which the width exceeds one page), headings can only appear in the first panel. Thus, the preceding calculations deal with the total width of the columns in the first panel rather than the total width of all the columns in the report.

Justifying a Column Title

How to:

Justify a Column Title Using a StyleSheet

You can left justify, right justify, or center a column title for a display field, BY field, ACROSS field, or calculated value using a StyleSheet.

If a title is specified with an AS phrase in a request, or with the TITLE attribute in a Master File, that title will be justified, as specified for the field in StyleSheet syntax, if such syntax exists in the request.

Justification behavior in HTML and PDF. For HTML reports, justification is implemented with respect to the report's width. That means a centered column title is centered over a report column. In contrast, for PDF reports the default justification area is the page width, rather than the report width. This results in column titles that are not centered over the report column. You can achieve justification based on report width in a PDF report by adding the command SET SQUEEZE=ON to your request. This command improves the appearance of the report by eliminating excessive white space between columns and implements justification over the report content.

Syntax: How to Justify a Column Title Using a StyleSheet

To justify a column title for a vertical sort column (generated by BY) or a display column (generated by PRINT, LIST, SUM, or COUNT), the StyleSheet syntax is

```
TYPE=TITLE, [COLUMN=column,] JUSTIFY=option, $
```

To justify a horizontal sort column title (generated by ACROSS), the StyleSheet syntax is

```
TYPE=ACROSSTITLE, [ACROSS=column,] JUSTIFY=option, $
```

To justify an ACROSS value or a ROW-TOTAL column title in an HTML report, use

```
TYPE=ACROSSVALUE, [COLUMN=column,] JUSTIFY=option, $
```

where:

TITLE

Specifies a vertical sort (BY) title or a display field title.

column

Specifies the column whose title you wish to justify. If you omit this attribute and value, the formatting will be applied to all of the report's column titles.

ACROSSTITLE

Specifies a horizontal sort (ACROSS) title.

ACROSSVALUE

Specifies a horizontal sort (ACROSS) value or a ROW-TOTAL column title.

option

Is the type of justification. Valid values are:

LEFT which left justifies the column title. This value is the default for an alphanumeric field.

RIGHT which right justifies the column title. This value is the default for a numeric or date field.

CENTER which centers the column title. You cannot center an ACROSSTITLE in a PDF report.

Example: Using a StyleSheet to Justify Column Titles for Display and BY Fields

This request identifies each column by its field name and justifies each title separately.

```
TABLE FILE GGSTORES
PRINT STORE_NAME STATE AS 'St' BY ADDRESS1
WHERE STATE EQ 'CA'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET SQUEEZE ON
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE SHEET *
TYPE = REPORT, GRID=OFF, $
TYPE=TITLE, COLUMN=STORE_NAME, JUSTIFY=CENTER, $
TYPE=TITLE, COLUMN=STATE, JUSTIFY=RIGHT, $
TYPE=TITLE, COLUMN=ADDRESS1, JUSTIFY=CENTER, $
ENDSTYLE
END
```

The output is:

```

      Contact           Store Name           St
JEFF DARFELL GOTHAM GRINDS #1244 CA
PAT SMILEY   GOTHAM GRINDS #1040 CA

```

Example: Using a StyleSheet to Justify a Column Title for ACROSS and ROW-TOTAL Fields

This request centers the column title, State, created by the ACROSS phrase over the two values (MT and WY) and the row total column title, Total by Gender, over the two row totals (Male Population and Female Population). Notice that each across value functions as a title for one or more columns in the report.

```

SET SQUEEZE = ON
TABLE FILE GGDEMOG
SUM MALEPOP98 FEMPOP98
ROW-TOTAL/D12 AS 'Total by Gender'
ACROSS ST
WHERE ST EQ 'WY' OR 'MT';
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=REPORT, GRID=OFF, $
TYPE=ACROSSTITLE, JUSTIFY=CENTER, FONT='TIMES', SIZE=11, STYLE=BOLD, $
TYPE=ACROSSVALUE, COLUMN=N5, JUSTIFY=CENTER, $
ENDSTYLE
END

```

The output is:

		State				Total by Gender	
MT		WY					
Male Population	Female Population	Male Population	Female Population	Male Population	Female Population	Male Population	Female Population
426357	438836	245023	245897	671380	684733		

Example: Using a StyleSheet to Justify a Column Title for a Calculated Value

This request identifies the column title of the calculated value and left justifies it over the data.

```
TABLE FILE SALES
SUM UNIT_SOLD RETAIL_PRICE
COMPUTE REV/D12.2M = UNIT_SOLD * RETAIL_PRICE;
BY PROD_CODE
WHERE CITY EQ 'NEW YORK'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=REPORT, GRID=OFF, $
TYPE=TITLE, COLUMN=REV, STYLE=BOLD, JUSTIFY=LEFT, $
ENDSTYLE
END
```

The output is:

<u>PROD CODE</u>	<u>UNIT SOLD</u>	<u>RETAIL PRICE</u>	<u>REV</u>
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
E1	30	\$.89	\$26.70
E3	35	\$1.09	\$38.15

Note: To run this report in PDF format, add the code ON TABLE SET SQUEEZE ON to eliminate excessive white space between columns and to justify column titles properly over the data.

Justifying a Label for a Subtotal or Grand Total

Although you cannot directly justify a customized label for a subtotal, if columns are being totaled or subtotaled by the one subtotal command, and you do not specify a column in the StyleSheet, formatting is applied to the totals and subtotals of all columns and to the labeling text that introduces the total and subtotal values.

Example: Justifying Subtotal and Grand Total Labels

This request subtotals the numeric columns in the report and right-justifies the output, including the text of the label that precedes the values for the subtotals. In this example, since numeric output is right justified by default, the justification specifications in the StyleSheet are used to reposition the labels. The default label for the automatically generated grand total is also right-justified.

```
TABLE FILE EMPLOYEE
SUM DED_AMT BY DED_CODE BY DEPARTMENT
BY BANK_ACCT
WHERE DED_CODE EQ 'CITY'
WHERE BANK_ACCT NE 0
ON DEPARTMENT SUBTOTAL AS 'Total City Deduction for'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=REPORT, GRID=OFF, $
TYPE=SUBTOTAL, STYLE=BOLD, JUSTIFY=RIGHT,$
TYPE=GRANDTOTAL, STYLE=BOLD, JUSTIFY=RIGHT,$
ENDSTYLE
END
```

The output is:

<u>DED_CODE</u>	<u>DEPARTMENT</u>	<u>BANK ACCT</u>	<u>DED AMT</u>
CITY	MIS	40950036	\$14.01
		122850108	\$31.76
		163800144	\$82.69
	Total City Deduction for MIS		\$128.46
	PRODUCTION	160633	\$7.42
		136500120	\$18.26
		819000702	\$60.24
	Total City Deduction for PRODUCTION		\$85.92
	TOTAL		\$214.38

Aligning Heading and Footing Elements

In this section:

- Aligning a Heading or Footing Element in an HTML Report
- Aligning a Heading or Footing Element Across Columns in an HTML Report
- Aligning Content in a Multi-Line Heading or Footing
- Aligning Decimals in a Multi-Line Heading or Footing
- Combining Column and Line Formatting in Headings and Footings

To align text and data in headings and footings based on factors other than left, right, and center justification, consider the following descriptions before deciding which alignment method best suits your needs.

Alignment Method	Applies to ...	When to use...	Related Methods
<p>1) StyleSheet Attributes:</p> <p>HEADALIGN</p> <p>COLSPAN</p> <p>JUSTIFY</p> <p>Details: See Aligning a Heading or Footing Element in an HTML Report on page 744.</p>	<p>HTML</p> <p>EXL2K</p>	<p>To align heading or footing items in HTML and EXL2K reports: If you expect to display reports in HTML or EXL2K format, use HEADALIGN options to align heading and footing items with either columns in the HTML table for the body of the report or with cells in an embedded HTML table. The browser handles alignment based on your specifications, without requiring unit measurements, which are required with WIDTH and JUSTIFY.</p> <p>To specify a heading or footing item that spans multiple columns: You can combine HEADALIGN syntax with the COLSPAN attribute to achieve this result. For details, see Aligning a Heading or Footing Element Across Columns in an HTML Report on page 747.</p>	

Alignment Method	Applies to ...	When to use...	Related Methods
<p>2) StyleSheet Attributes:</p> <p>WIDTH</p> <p>JUSTIFY</p> <p>Details: See Aligning Content in a Multi-Line Heading or Footing on page 755.</p>	<p>HTML</p> <p>PDF</p> <p>PS</p>	<p>For portability between HTML and PDF: To code a request that can be used without revision to produce identical output in HTML (with internal Cascading Style Sheets) and in PDF, use WIDTH and JUSTIFY attributes in your StyleSheet. These settings can be applied to report, page, and sort headings and footings.</p> <p>To align heading or footing items: Used together, WIDTH and JUSTIFY allow you to align specific items in the heading, rather than entire headings or footings or entire heading or footing lines, where the implied justification width is the total width of the report panel. To right- or center-justify an item in a heading or footing, you must know the width of the area you want to justify it in. That information is provided by the WIDTH attribute.</p> <p>To align decimal points in a multi-line heading or footing: Use this technique to align decimal points in data that has varying numbers of decimal places. Define the width of the decimal item, then measure how far in from the right side of a column you want to position the decimal point. This places the decimal point in the same position in a column, regardless of the number of decimal places displayed to its right.</p>	<p>For an HTML or EXL2K report, you can align specific items with HEADALIGN options.</p>

Alignment Method	Applies to ...	When to use...	Related Methods
3) StyleSheet Attribute: POSITION Details: See Positioning a Report Component on page 706.	PDF PS HTML (limited)	<p>To set starting positions for headings or footings, or items within them: Use POSITION syntax to specify absolute and relative starting positions.</p> <p>In HTML, with an internal Cascading Style Sheet, you can use POSITION to specify the starting point for a heading or footing line. You can also position an image in a heading or footing.</p> <p>To align heading and footing items with columns: Use POSITION syntax to align a heading item with a column position. For example, the syntax</p> <pre>TYPE=SUBHEAD, LINE=1, ITEM=3, POSITION=SALES, \$</pre> <p>places ITEM 3 of the sort heading at the horizontal position where the column SALES is.</p>	<p>For a PDF report, you can accomplish most positioning with WIDTH and JUSTIFY.</p> <p>For an HTML report, you can align a heading item with a column by setting the HEADALIGN attribute to BODY.</p>

Aligning a Heading or Footing Element in an HTML Report

How to:

Align a Heading or Footing Element in an HTML Report

For HTML output (and for Excel 2000 output, which uses HTML alignment), you can position text and field items in headings and footings using HEADALIGN options. These options work within the limitations of HTML and browser technologies to provide a significant degree of formatting flexibility. Here is how HEADALIGN works.

When HEADALIGN is set either to BODY or INTERNAL, output is laid out as an HTML table, which means that the browser determines the widths of the columns, thereby limiting the precise positioning of items. A basic rule governs the placement of heading or footing items: each item (text or embedded field) is placed in sequence into the next HTML table cell (<TD>). When HEADALIGN is set to NONE, the default, all the items in the heading or footing are strung together, inside a single cell. The browser stretches the heading table and the report table to accommodate the length of the text.

You can exercise control over the placement of items by overriding the default and choosing either BODY or INTERNAL:

- ❑ HEADALIGN=BODY puts heading item cells in the same HTML table as the body of the report, ensuring that the items in the heading and the data in the body of the report line up naturally since they have the same column widths. This is a simple and useful way to align heading items with columns of data. For example, suppose that you have computed subtotal values that you want to include in a sort footing. Using HEADALIGN=BODY, you can align the subtotals in the same columns as the data that is being totaled.
- ❑ HEADALIGN=INTERNAL puts the heading items in an HTML table of its own. This allows the heading items to be aligned vertically with each other, independent of the data, since the widths of the heading items do not affect the width of the report columns and vice versa.

To break a text string into multiple parts for manipulation across columns in an HTML table, you can use `<+0>` spot markers in the request.

You can use HEADALIGN options in conjunction with the COLSPAN attribute. COLSPAN allows heading items to span multiple table columns, thereby providing additional flexibility in how you can design your headings. For details, see [Aligning a Heading or Footing Element in an HTML Report](#) on page 744.

If there is more than one heading or footing type in a report, you can individually align any element within each of them using this syntax.

HEADALIGN is not supported for PDF reports. For alignment methods supported by both HTML and PDF, see [Aligning Content in a Multi-Line Heading or Footing](#) on page 755.

Syntax: **How to Align a Heading or Footing Element in an HTML Report**

```
TYPE = {REPORT|headfoot}, HEADALIGN = option, $
```

where:

REPORT

Applies the chosen alignment to all heading and footing elements in a report.

headfoot

Is the type of heading or footing. Valid values are TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD, and SUBFOOT.

option

Is the type of alignment. Valid values are:

NONE which places heading items in an embedded HTML table inside the main (body) table, and strings together, in a single cell of the embedded table, all the heading items (text and fields) on a line. This value is the default.

INTERNAL which places heading items in an HTML table of its own, with each item in a separate cell. This allows the heading items to be aligned vertically with each other, independent of the data columns. The widths of the heading items do not affect the widths of the report columns and vice versa.

BODY which aligns heading items with data columns by placing the items in the cells of the same HTML table as the body of the report. Since they have the same column widths, the items in the heading and the data in the body of the report line up naturally.

Example: Aligning Elements in a Page Heading Using a Separate HTML Table

This request creates an embedded HTML table for a page heading, within the HTML table that governs alignment in the body of the report. This table has three rows and three columns to accommodate all the heading elements.

In the first line of the heading, a spot marker (<+0>) creates two text elements: the first element is blank, and the second element is Gotham Grinds, Inc. In the output, the second element appears in the second cell of the first row of the embedded table.

The second and fourth lines of the heading are blank.

The spot markers in the third line of the heading split it into three text elements: Orders Report, blank, Run on: &DATE. In the output, each element appears in a cell in the third row of the embedded HTML table, in the order specified in the request.

```
TABLE FILE GGORDER
HEADING
" <+0>Gotham Grinds, Inc."
" "
"Orders Report <+0> <+0> Run on: &DATE"
" "
PRINT ORDER_NUMBER ORDER_DATE STORE_CODE QUANTITY
BY PRODUCT_CODE BY PRODUCT_DESCRIPTION
IF RECORDLIMIT EQ 10
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE = REPORT, GRID = ON, $
TYPE = HEADING, HEADALIGN = INTERNAL, STYLE = BOLD, $
ENDSTYLE
END
```

GRID=ON in the request enables you to see the embedded HTML table for the heading, and the main HTML table for the body of the report. The positioning is maintained when the grid lines are hidden (off).

The output is:

Gotham Grinds, Inc.					
Orders Report			Run on: 06/19/02		
Product Code	Product	Order Number	Order Date	Store Code	Ordered Units
B141	Hazelnut	1	01/01/96	R1019	300
		2	01/01/96	R1019	117
B142	French Roast	3	01/01/96	R1019	126
		4	01/01/96	R1019	86
		5	01/01/96	R1019	49
G100	Mug	6	01/01/96	R1019	289
		7	01/01/96	R1019	231
G104	Thermos	8	01/01/96	R1019	347
G110	Coffee Grinder	9	01/01/96	R1019	265
G121	Coffee Pot	10	01/01/96	R1019	309

Aligning a Heading or Footing Element Across Columns in an HTML Report

How to:

Align a Heading or Footing Element Across Columns

With HEADALIGN=BODY, each heading or footing element is aligned with a data column in an HTML report; with HEADALIGN=INTERNAL, each element is continued in a column of an HTML table created and aligned specifically for the report heading or footing. By default, every heading or footing element (ITEM) is placed in the first available column. However, you can position an item to span multiple columns using the COLSPAN attribute. For details about HEADALIGN options, see [Aligning a Heading or Footing Element in an HTML Report](#) on page 744.

You must specify the HEADALIGN and COLSPAN attributes in two separate StyleSheet declarations, since HEADALIGN applies to an entire heading or footing, while COLSPAN applies to a specific item in a heading or footing. This feature is supported for HTML only. PDF display format does not support the COLSPAN attribute.

Syntax: **How to Align a Heading or Footing Element Across Columns**

TYPE = *headfoot*, [*subtype*,] COLSPAN = *n*, \$

where:

headfoot

Is the type of heading or footing. Valid values are TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD, and SUBFOOT.

subtype

Are additional attributes that identify the report component. These options can be used separately or in combination, depending upon the degree of specificity required to identify an element. Valid values are:

LINE which identifies a line by its position in a heading or footing. Identifying individual lines enables you to format each line differently.

If a heading or footing has multiple lines and you apply a StyleSheet declaration that does not specify LINE, the declaration is applied to *all* lines. Blank lines are counted when interpreting the value of LINE.

OBJECT which identifies an element in a heading or footing as a text string or field value. Valid values are TEXT or FIELD. TEXT may represent free text or a Dialogue Manager ampersand (&) variable.

It is not necessary to specify OBJECT=TEXT unless you are styling both text strings and embedded fields in the same heading or footing.

ITEM which identifies an item by its position in a line. To divide a heading or footing line into items, you can use the <+0> spot marker.

To determine the ITEM for an OBJECT, follow these guidelines:

- ❑ When used with OBJECT=TEXT, count only the text strings from left to right.
- ❑ When used with OBJECT=FIELD, count only values from left to right.
- ❑ When used without OBJECT, count text strings and field values from left to right.

If you apply a StyleSheet declaration that specifies ITEM, the number is counted from the beginning of each line in the heading or footing, not just from the beginning of the first line.

COLSPAN

Is an attribute that aligns an item in the width spanned by multiple columns.

n

Is the column with which the specified item is aligned.

Example: Comparing Output Generated With HEADALIGN Options

The requests that follow illustrate the differences in alignment with each HEADALIGN setting. The grid lines are exposed in the output to help distinguish the HTML table created for the body of the report from the embedded HTML tables created for the heading in some variations.

All HEADALIGN settings are compatible with COLSPAN syntax, which allows heading items to span multiple columns.

```
TABLE FILE CAR
SUM SALES BY COUNTRY BY CAR BY MODEL
ON COUNTRY SUBHEAD
"This is my subhead"
" "
"Country is:<COUNTRY Car is:<CAR"
"Model is:<MODEL"
IF COUNTRY EQ 'ENGLAND'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=SUBHEAD, HEADALIGN=OPTION, $
TYPE=SUBHEAD, LINE=1, ITEM=1, COLSPAN=4, JUSTIFY=CENTER, $
ENDSTYLE
END
```

HEADALIGN=NONE creates a separate table with default left alignment. The text and fields in each heading line are strung together in a single HTML table cell. In order to get this result, do not include the TYPE=SUBHEAD line that contains the COLSPAN=R, JUSTIFY=CENTER attributes in the StyleSheet:

TYPE=SUBHEAD, HEADALIGN=NONE, \$

COUNTRY	CAR	MODEL	SALES
This is my subhead			
Country is:ENGLAND Car is:JAGUAR Model is:V12XKE AUTO			
ENGLAND	JAGUAR	V12XKE AUTO	0
		XJ12L AUTO	12000
	JENSEN	INTERCEPTOR III	0
	TRIUMPH	TR7	0

HEADALIGN=NONE with COLSPAN

TYPE=SUBHEAD, HEADALIGN=NONE, \$

TYPE=SUBHEAD, LINE=1, ITEM=1, COLSPAN=4, JUSTIFY=CENTER, \$

COUNTRY	CAR	MODEL	SALES
This is my subhead			
Country is:ENGLAND Car is:JAGUAR Model is:V12XKE AUTO			
ENGLAND	JAGUAR	V12XKE AUTO	0
		XJ12L AUTO	12000
	JENSEN	INTERCEPTOR III	0
	TRIUMPH	TR7	0

The first line is centered across all four columns of the internal table, based on the COLSPAN=4 setting.

HEADALIGN=INTERNAL creates a separate HTML table. Columns are generated based on the number of items (text and fields) in the heading; each item is placed in a separate cell. These columns do *not* correspond to those in the HTML table for the report body.

TYPE=SUBHEAD, HEADALIGN=INTERNAL, \$

COUNTRY	CAR	MODEL	SALES
This is my subhead			
Country is:	ENGLAND	Car is:	JAGUAR
Model is:	V12XKE AUTO		
ENGLAND	JAGUAR	V12XKE AUTO	0
		XJ12L AUTO	12000
	JENSEN	INTERCEPTOR III	0
	TRIUMPH	TR7	0

Country is aligned with Model in the first column of the internal table. The value of <COUNTRY is aligned with the value of <MODEL in the second column.

HEADALIGN=INTERNAL with COLSPAN

TYPE=SUBHEAD, HEADALIGN=INTERNAL, \$

TYPE=SUBHEAD, LINE=1, ITEM=1, COLSPAN=4, JUSTIFY=CENTER, \$

COUNTRY	CAR	MODEL	SALES
This is my subhead			
Country is:	ENGLAND	Car is:	JAGUAR
Model is:	V12XKE AUTO		
ENGLAND	JAGUAR	V12XKE AUTO	0
		XJ12L AUTO	12000
	JENSEN	INTERCEPTOR III	0
	TRIUMPH	TR7	0

The first line is centered across all 4 columns of the internal table, based on the COLSPAN=4 setting.

HEADALIGN=BODY places the heading lines within the cells of the main HTML table. As a result, the columns of the heading correspond to the columns of the main table.

TYPE=SUBHEAD, HEADALIGN=BODY, \$

COUNTRY	CAR	MODEL	SALES
This is my subhead			
Country is:	ENGLAND	Car is:	JAGUAR
Model is:	V12XKE AUTO		
ENGLAND	JAGUAR	V12XKE AUTO	0
		XJ12L AUTO	12000
	JENSEN	INTERCEPTOR III	0
	TRIUMPH	TR7	0

Country is aligned with Model in the first column of the main (body) HTML table. The value of <COUNTRY is aligned with the value of <MODEL in the second column.

HEADALIGN=BODY with COLSPAN

TYPE=SUBHEAD, HEADALIGN=BODY, \$

TYPE=SUBHEAD, LINE=1, ITEM=1, COLSPAN=4, JUSTIFY=CENTER, \$

COLSPAN controls the cross-column alignment of the first row of the heading.

COUNTRY	CAR	MODEL	SALES
This is my subhead			
Country is:	ENGLAND	Car is:	JAGUAR
Model is:	V12XKE AUTO		
ENGLAND	JAGUAR	V12XKE AUTO	0
		XJ12L AUTO	12000
	JENSEN	INTERCEPTOR III	0
	TRIUMPH	TR7	0

Example: Aligning and Styling a Text Field in a Sort Footing

This example uses the following Master File and MODIFY procedure to create a data source with a text field:

Master File:

```
FILENAME = TXTFLD, SUFFIX = FOC,$
SEGNAME=TEXTSEG, SEGTYPE = S1,$
  FIELDNAME = CATALOG, FORMAT = A10, $
  FIELDNAME = TEXTFLD,      FORMAT = TX50,$
```

MODIFY Procedure to create the TXTFLD data source:

```
CREATE FILE TXTFLD
MODIFY FILE TXTFLD
FIXFORM CATALOG/10 TEXTFLD
DATA
COURSE100 This course provides the junior programmer
with the skills needed to code simple reports.%$
COURSE200 This course provides the advanced programmer with
techniques helpful in developing complex
applications.%$
END
```

This request applies boldface type to the second line of a multiple-line sort footing, which includes the text Course Description as well as the text of the field TEXTFLD. Line 1 of the sort footing is the text Evening Course.

```
TABLE FILE TXTFLD
BY CATALOGA SUBFOOT
"Evening Course"
"Course Description: <TEXTFLD"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE SHEET *
TYPE = REPORT, GRID = OFF, $
TYPE = SUBFOOT, HEADALIGN = BODY, $
TYPE = SUBFOOT, LINE = 2, STYLE = BOLD, $
ENDSTYLE
END
```

The output is:

CATALOG

COURSE100

Evening Course

Course Description: This course provides the junior programmer with the skills needed to code simple reports.

COURSE200

Evening Course

Course Description: This course provides the advanced programmer with techniques helpful in developing complex applications.

If the StyleSheet instead identifies the text field as an object for styling

```
TYPE = SUBFOOT, HEADALIGN = BODY, $
```

```
TYPE = SUBFOOT, LINE = 2, OBJECT = FIELD, STYLE = BOLD, $
```

then only the text in TEXTFLD is bold:

CATALOG

COURSE 100

Evening Course

Course Description: This course provides the junior programmer with the skills needed to code simple reports.

COURSE 200

Evening Course

Course Description: This course provides the advanced programmer with techniques helpful in developing complex applications.

Aligning Content in a Multi-Line Heading or Footing

How to:

Align Heading Text and Data in Columns

Reference:

Line and Item Formatting in a Multi-Line Heading or Footing

The HEADALIGN and COLSPAN syntax described in [Aligning a Heading or Footing Element in an HTML Report](#) on page 744 is specific to HTML reports. This topic describes how you can design reports that are printable across HTML and PDF formats. Using the WIDTH and JUSTIFY syntax in a StyleSheet, you can:

- ❑ Align vertical sets of text or data as columnar units. .
- ❑ Combine columnar formatting with line-by-line formatting. See [Combining Column and Line Formatting in Headings and Footings](#) on page 762.
- ❑ Align decimal points when the data displayed has varying numbers of decimal places. See [Aligning Decimals in a Multi-Line Heading or Footing](#) on page 760.

You can apply WIDTH and JUSTIFY attributes to report headings and footings, page headings and footings, and sort headings and footings, using either mono-space or proportional fonts.

These techniques rely on internal Cascading Style Sheets, which support FOCUS StyleSheet attributes that were not previously available for HTML reports. The syntax associated with these techniques resolves the problem of having to format headings differently for HTML reports (using HEADALIGN and COLSPAN) and PDF and PS reports (using POSITION and spot markers).

While the WIDTH and JUSTIFY attributes are particularly useful when you need to format a multi-line heading or footing, or align stacked decimals, you can also use this syntax to position items in an individual heading or footing line.

Syntax: **How to Align Heading Text and Data in Columns**

For a multi-line report or page heading or footing, use the syntax:

```
TYPE=headfoot, WRAP=OFF, $
TYPE=headfoot, [LINE=line_#,,] ITEM=item_#, [OBJECT={TEXT|FIELD}],
  WIDTH=width, [JUSTIFY=option,] $
```

For a multi-line sort heading or footing, use the syntax

```
TYPE=headfoot, WRAP=OFF, $
TYPE={SUBHEAD|SUBFOOT}, [BY=sortfield] [LINE=line_#,,] ITEM=item_#,
  [OBJECT={TEXT|FIELD}], WIDTH=width, [JUSTIFY=option,] $
```

where:

headfoot

Is the type of heading or footing. Valid values are TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD, and SUBFOOT.

sortfield

When TYPE=SUBHEAD or SUBFOOT, you can specify alignment for the sort heading or sort footing associated with a particular sort field. If no sort field is specified, formatting is applied to the sort headings or footings associated with all sort fields.

LINE

Is an optional entry that identifies a line by its position in a heading or footing. Identifying individual lines enables you to format each one differently.

If a heading or footing has multiple lines and you apply a StyleSheet declaration that does not specify LINE, the declaration is applied to *all* lines. Blank lines are counted when interpreting the value of LINE.

You can use LINE in combination with ITEM.

ITEM

Is a required entry when you are using WIDTH to control alignment. An item can identify either:

- A vertical set of text or data that you wish to align as a columnar unit. You must identify each vertical unit as an item.
- An item's position in a line. You must identify each line element as an item. See [Line and Item Formatting in a Multi-Line Heading or Footing](#) on page 757 for information about acceptable variations.

You can use either or both approaches for a single heading or footing.

To divide a heading or footing line into items, you can use the <+0> spot marker. The number of items you can identify is limited by the cumulative widths of the items in the heading or footing, within the physical boundaries of the report page.

You can use ITEM in conjunction with OBJECT to refine the identification of an element whose width you want to define. To determine the ITEM for an OBJECT, follow these guidelines:

- When used with OBJECT=TEXT, count only the text strings from left to right.
- When used with OBJECT=FIELD, count only values from left to right.
- When used without OBJECT, count text strings and field values from left to right.

If you apply a StyleSheet declaration that specifies ITEM, the number is counted from the beginning of each line in the heading or footing, not just from the beginning of the first line

OBJECT

Is an optional entry that identifies an element in a heading or footing as a text string or field value. Valid values are TEXT or FIELD. TEXT may represent free text or a Dialogue Manager ampersand (&) variable.

It is not necessary to specify OBJECT=TEXT unless you are styling both text strings and embedded fields in the same heading or footing.

width

Is the measurement expressed in units (inches by default), which is required to accommodate the longest text string or field value associated with a numbered item. For details, see [How to Measure for Column Width and Decimal Alignment](#) on page 761.

option

Is the type of justification. Valid values are:

[LEFT](#) which left justifies the heading or footing. This value is the default.

[RIGHT](#) which right justifies the heading or footing.

[CENTER](#) which centers the heading or footing.

DECIMAL (*n*)

Is the measurement expressed in units (inches by default), which specifies how far in from the right side of a column to place the decimal point. With this specification, you can locate the decimal point in the same position within a column, regardless of the number of decimal places displayed to its right.

The measurement will be a portion of the width specified for this item. For details, see [How to Measure for Column Width and Decimal Alignment](#) on page 761.

Reference: Line and Item Formatting in a Multi-Line Heading or Footing

Line formatting maximizes your control over the items you identify on each line:

- ❑ You can align and stack the same number of items with uniform widths. For example,

Line 1	Item 1	item 2	Item 3
Line 2	Item 1	Item 2	Item 3

- ❑ You can also align different numbers of items as long as the items on each line have the same starting point and the same cumulative width.

Line 1	Item 1	item 2	
Line 2	Item 1	Item 2	Item 3

Do not use HEADALIGN or COLSPAN syntax, which are specific to HTML reports and may conflict with WIDTH and JUSTIFY settings.

For HTML reports, turn WRAP OFF (ON is the default) to ensure proper processing of WIDTH and JUSTIFY.

Example: Aligning Data and Text in a Multi-Line Heading or Footing

In the following free-form report, content is defined entirely in the sort heading, where text and data are stacked to support comparison among manufacturing plants. Each set of data is aligned vertically, to appear as a column. To achieve this affect, each vertical unit is identified as an item: the first column of text is *item 1*; the next column of data is *item 2*, and so on.

Note especially the last column, in which decimal data with different numbers of decimal places is lined up on the decimal point to facilitate reading and comparison.

Plant:	BOS	Quantity:	1,033,818
Order Date:	2002/01/02	Cost of Goods:	173,484,823.50000
Store Code:	1003MD	Line Total:	\$262,433,960.63
Plant:	DAL	Quantity:	390,844
Order Date:	2002/01/02	Cost of Goods:	64,880,802.75000
Store Code:	2011TX	Line Total:	\$97,751,846.65
Plant:	LA	Quantity:	229,256
Order Date:	2002/01/02	Cost of Goods:	37,788,489.00000
Store Code:	999999	Line Total:	\$57,507,080.82
Plant:	ORL	Quantity:	386,909
Order Date:	2002/01/02	Cost of Goods:	64,702,888.50000
Store Code:	3002FL	Line Total:	\$97,616,855.94
Plant:	SEA	Quantity:	86,680
Order Date:	2002/01/02	Cost of Goods:	15,066,598.50000
Store Code:	1003WA	Line Total:	\$22,742,743.88
Plant:	STL	Quantity:	776,743
Order Date:	2002/01/02	Cost of Goods:	129,000,075.75000
Store Code:	3002IL	Line Total:	\$195,970,536.09

The chart below breaks out the structure of the previous report:

Item 1: Text	Item 2: Data values	Item 3: Text	Item 4: Values with decimal places
Plant	BOSDAL, and so on	Quantity	<i>n,nnn,nnn</i>
Order Date	2002/01/02	Cost of Goods	<i>nnn,nnn,nnn.ddddd</i>
Store Code	1003MD	Line Total	<i>\$nnn,nnn,nnn.dd</i>

For each item, you specify the width of the column and the justification of its content, as illustrated in the following code.

```
DEFINE FILE CENTORD
COST/D20.5 = LINE_COGS * .75 ;
END

TABLE FILE CENTORD
BY PLANT NOPRINT SUBHEAD
"Plant: <PLANT Quantity: <QUANTITY"
"Order Date: <ORDER_DATE Cost of Goods: <COST"
"Store Code: <STORE_CODE Line Total: <LINEPRICE"
ON TABLE SET PAGE-NUM OFF
ON TABLE SET HTMLCSS ON
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE=REPORT, FONT='TIMES', $
TYPE=REPORT, GRID=OFF, $
TYPE=SUBHEAD, ITEM=1, WIDTH=1.00, JUSTIFY=RIGHT, $
TYPE=SUBHEAD, ITEM=2, WIDTH=1.25, JUSTIFY=RIGHT, $
TYPE=SUBHEAD, ITEM=3, WIDTH=1.25, JUSTIFY=RIGHT,$
TYPE=SUBHEAD, ITEM=4, WIDTH=2.0, JUSTIFY=DECIMAL(.6),$
ENDSTYLE
END
```

This procedure produces a three-line sort heading, broken out as four items, each with a measured width and defined justification. The decimal item (4) uses a variation on standard justification to line up the decimal points. For details, see [How to Align Heading Text and Data in Columns](#) on page 755 and [Aligning Decimals in a Multi-Line Heading or Footing](#) on page 760.

Note: To take advantage of this feature for an HTML report, turn on internal Cascading Style Sheets (SET HTMLCSS=ON). This command enables FOCUS StyleSheet attributes that were not previously available for HTML reports. This line of code is ignored for a PDF report.

Aligning Decimals in a Multi-Line Heading or Footing

How to:

Measure for Column Width and Decimal Alignment

The ability to align heading content in a multi-line heading based on width and justification values has special benefit in reports that contain data with different numbers of decimal places. For example, if a figure is in dollars, it is formatted with a decimal point and two places for zeroes; if in Swiss francs, it is formatted with a decimal place and four zeroes; if in yen, the decimal is at the end with no zeroes. In addition, sometimes the currency or units do not vary, but the number of digits of decimal precision varies.

By aligning the decimal points in a vertical stack, you can more easily read and compare these numbers, as illustrated in the following output:

Floating decimal points		Aligned decimal points	
Bond ----- Galosh Ltd. Mukluk Inc. Overshoe Inc.	Face Value ----- 22375.5784596 1212345.457 232.45484	Bond ----- Galosh Ltd. Mukluk Inc. Overshoe Inc.	Face Value ----- 22375.5784596 1212345.457 232.45484

The technique uses a width specification for the item that contains decimals, combined with a variation on standard left/right/center justification to achieve the proper decimal alignment. For the syntax that generates this output, see [How to Align Heading Text and Data in Columns](#) on page 755.

Procedure: How to Measure for Column Width and Decimal Alignment

Measuring Width. Determining the width of a heading or footing item is a three-step process:

1. Identify the maximum number of characters in a text string or field.
2. For a text string, simply count the characters. For a field, refer to the format specification in the Master File or in a command such as a DEFINE.
3. Measure the physical space in units (for example, in inches) that is required to display the number of characters identified in step 1, based on the size of the font you are using. For example, the following value of the COUNTRY field would measure as follows:

Font	Font size	Comparison	Inches
Helvetica	10	England	.5
Times New Roman	10	England	.44
Courier	10	England	.56

Tip: Consider using a consistent set of fonts in your reports to make your measurements reusable.

Measuring for Decimal Alignment. After you have determined the width of an item, you can do a related measurement to determine the physical space required to display decimal data with a varying number of digits to the right of the decimal point.

1. Determine the maximum number of decimal places you need to accommodate to the right of the decimal place, plus the decimal point itself.
2. Measure the physical space in units (for example, in inches) that is required to display the number of characters identified in step 1, based on the size of the font you are using.

Combining Column and Line Formatting in Headings and Footings

By combining column and line formatting, you can create complex reports in which different ranges of lines in the same heading or footing have different numbers of aligned columns in different locations.

Example: Combining Column and Line Formatting to Align Items in a Sort Heading

This request produces a free-form report in which content is defined in a seven-line sort heading. Data is stacked in two groupings:

- ❑ The first grouping identifies the regions and state.
- ❑ The second grouping provides other information for each region/state pair.

Although this is a single sort heading, our goal is to format the information in each grouping a bit differently to provide emphasis and facilitate comparison. The request also demonstrates a coding technique that makes formatting changes easier for the report designer. See the annotations following the code for details.

As you review the sample request, keep in mind that a heading can contain two kinds of items: text and embedded fields. A text item consists of any characters, even a single blank, between embedded fields and/or spot markers. In particular, if you have a single run of text that you want to treat as two items, you can separate the two items using a `<+0>` spot marker. For example, in the heading line:

```
" <+0>Region:<REGION"
```

item #1 is a single blank space.

item #2, separated by the `<+0>` spot marker, is the text Region:

item #3 is the embedded field `<REGION`.

Request and annotations:

```

TABLE FILE CENTORD
BY REGION NOPRINT SUBHEAD
1. " <+0>Region:<REGION"
2. " <+0>State :<STATE"
   " "
3. "Product Number:<PROD_NUM <+0>Quantity:<QUANTITY"
4. "Product Type:<PRODCAT <+0>Price:<PRICE"
5. "Product Category:<PRODTYPE <+0>Cost:<COST"
   " "
ON TABLE SET PAGE-NUM OFF
6. ON TABLE SET HTMLCSS ON
ON TABLE HOLD FORMAT HTML AS NF958055
ON TABLE SET STYLESHEET *
TYPE=REPORT, FONT='TIMES', $
TYPE=REPORT, GRID=OFF, $
-* Bottom section of subhead:
7. TYPE=SUBHEAD, ITEM=1, WIDTH=1.25, JUSTIFY=RIGHT, $
8. TYPE=SUBHEAD, ITEM=2, WIDTH=1.00, JUSTIFY=RIGHT, $
9. TYPE=SUBHEAD, ITEM=3, WIDTH=1.0, $
10. TYPE=SUBHEAD, ITEM=4, WIDTH=1.00, JUSTIFY=RIGHT,$
11. TYPE=SUBHEAD, ITEM=5, WIDTH=1.5, JUSTIFY=DECIMAL(.6),$
    -* Top section of subhead (overrides above ITEM defaults
    -*   for lines 1 and 2):
12. -SET &INDENT = 1.5;
13. TYPE=SUBHEAD, LINE=1, ITEM=1, WIDTH=&INDENT, $
14. TYPE=SUBHEAD, LINE=1, ITEM=2, WIDTH=1, JUSTIFY=LEFT, $
15. TYPE=SUBHEAD, LINE=1, ITEM=3, SIZE=14, WIDTH=2, JUSTIFY=LEFT, $
16. TYPE=SUBHEAD, LINE=2, ITEM=1, WIDTH=&INDENT, $
17. TYPE=SUBHEAD, LINE=2, ITEM=2, WIDTH=1, JUSTIFY=LEFT, $
18. TYPE=SUBHEAD, LINE=2, ITEM=3, WIDTH=1.25, JUSTIFY=LEFT, $
ENDSTYLE
END

```

Here is the output generated by this request. It highlights the key information and its relationship by aligning text and data, including decimal data in which decimal points are aligned for easy comparison.

```

                Region:      EAST
                State :      MD

Product Number:      1034           Quantity:      1,043,507
Product Type:      PDA Devices      Price:        5,991,793.00
Product Category:   Digital         Cost:         4,321,273.00
    
```

```

                Region:      NORTH
                State :      IL

Product Number:      1036           Quantity:      707,199
Product Type:      PDA Devices      Price:        1,484,680.00
Product Category:   Digital         Cost:         1,064,154.00
    
```

```

                Region:      SOUTH
                State :      FL

Product Number:      1034           Quantity:      827,871
Product Type:      PDA Devices      Price:        1,605,248.00
Product Category:   Digital         Cost:         1,152,895.00
    
```

```

                Region:      WEST
                State :      WA

Product Number:      1036           Quantity:      325,673
Product Type:      PDA Devices      Price:        636,438.00
Product Category:   Digital         Cost:         455,396.00
    
```

Line #	Description
1-2	Defines the content for the <i>top</i> , two-line section of the sort heading. Each line contains three items: the first is a blank area (denoted by a space, separated from the next item by a <+0> spot marker); the second contains text; the third contains data values related to the text.

Line #	Description
3-5	Defines the content for the <i>bottom</i> , three-line section of the sort heading. Each line contains five items: text; data values related to the text; a blank column (denoted by a space, separated from the next item by a null spot marker); text; data values related to the text.
6	Turns on internal Cascading Style Sheets, a requirement for these formatting options. This command enables FOCUS StyleSheet attributes that were not previously available for HTML reports. This line of code is ignored for a PDF report.
7-11	<p>Specifies the basic formatting characteristics for the sort heading by breaking the content into five columns, each identified as an item with a defined width, and justification information for all but the empty column.</p> <p>Important: Had additional formatting code (annotated as 12-17) not been included in the request, the specifications annotated as 7-11 would have applied to the entire sort heading—that is, the formatting of the three columns in the top section of the heading would have been based on the specifications for the first three columns described below. However, that is not the desired effect, so a second section of StyleSheet code is defined to override this formatting for lines 1 and 2 of the sort heading. See annotations 12-18.</p> <p>The formatting of the <i>bottom</i>, three-line section of the heading is controlled by the following specifications:</p> <p>Item 1 identifies a columnar unit that contains text (that is, Product Number, Product Type, Product Category; it has a defined width of 1.25 inches and the text is right justified.</p> <p>Item 2 identifies a columnar unit that contains data values related to the text in item 1; it has a defined width of 1 inch and the data is right justified.</p> <p>Item 3 identifies a columnar unit that contains blank space and serves as a separator between columns; it has a width of 1 inch. Justification is not relevant.</p> <p>Item 4 identifies a columnar unit that contains text (Quantity, Price, Cost); it has a defined width of 1 inch and the text is right justified.</p> <p>Item 5 identifies a columnar unit that contains a decimal value; the width of the column that contains the value is 1.5 inches, with the decimal point anchored .6 inches in from the right edge of that column.</p> <p>The common width and justification definitions enforce the proper alignment of each item.</p>

Line #	Description
<p>12</p>	<p>Defines a variable called &INDENT, with a width setting of 1.5 inches. This variable defines the width of the blank area (item 1) at the beginning of lines 1 and 2 of the sort heading.</p> <p>Defining the width as a variable enables you to experiment with different widths simply by changing the value in one location. For a complex report, this technique can potentially save development time.</p>
<p>13-18</p>	<p>Specifies line-by-line formatting for the <i>top</i>, two-line section of the sort heading. This code overrides the previous formatting for lines 1 and 2 of the sort heading because it specifies a line number.</p> <p>Item 1 on each line refers to the blank area. The width is defined as a variable and implemented based on the current value of &INDENT.</p> <p>Item 2 on each line refers to the text area; it has a defined width of 1 inch and the text is left justified.</p> <p>Item 3 on each line refers to the data values; it has a defined width of 2 inches and the data is left justified.</p> <p>The common width and justification definitions enforce the proper alignment of each item.</p> <p>Notice that item 1 in line 15 defines a font size for the data values associated with the REGION field. All other items on both lines use a default font. Line-by-line formatting enables you to define a unique characteristic for a single item.</p>

Adding Grids and Borders

How to:

- Control Grid Display in HTML Reports
- Add and Format Borders
- Add and Adjust Grid Lines (PDF or PS)

Reference:

- Grid Display Attributes

By default, an HTML report contains horizontal and vertical grid lines. You can remove the grid lines or adjust their use on a horizontal (BY) sort field. Grid characteristics apply to an entire HTML report, not to individual components of a report.

You can emphasize headings, footings, and column titles in a report by adding borders and grid lines around them. **Borders:** In an HTML, PDF, or PS report, you can use BORDER attributes in a StyleSheet to specify the weight, style, and color of border lines. If you wish, you can specify formatting variations for the top, bottom, left, and right borders.

Grids: In an HTML report, you can use the GRID attribute in a StyleSheet to turn grid lines on and off for the entire report. When used in conjunction with internal Cascading StyleSheets, GRID produces a thin grid line rather than a thick double line (the HTML default). In PDF reports you can use the HGRID and VGRID attributes to add horizontal or vertical grid lines and adjust their density.

Note: The SET GRID parameter, which applies to graphs, is not the same as the GRID StyleSheet attribute.

Reference: Grid Display Attributes

Attribute	Description	Applies to
GRID	Controls grid display.	HTML
HGRID	Controls horizontal grid display and grid line density.	PDFPS
VGRID	Control vertical grid display and grid line density.	PDFPS

Syntax: How to Control Grid Display in HTML Reports

```
[TYPE=REPORT,] GRID= option, $
```

where:

TYPE=REPORT

Applies the grid to the entire report. Not required, as it is the default.

option

Is one of the following:

ON applies a grid to a report. Does not apply grid lines to cells underneath a BY field value until the value changes. Column titles are not underlined. This value is the default.

OFF disables the default grid. Column titles are underlined. You can include blank lines and underlines. You cannot wrap cell data. With this setting, a report may be harder to read.

FILL applies grid lines to all cells of a report. Column titles are not underlined.

Syntax: How to Add and Format Borders

To request a uniform border, use this syntax:

```
TYPE=type, BORDER=option, [BORDER-STYLE=line_style,]  
[BORDER-COLOR={color|RGB(r g b)},] $
```

To specify different characteristics for the top, bottom, left, and/or right borders, use the syntax

```
TYPE=type, BORDER-position=option,  
[BORDER[-position]-STYLE=line_style,]  
[BORDER[-position]-COLOR={color|RGB(r g b)},] $
```

where:

type

Identifies the report component to which borders are applied.

option

Can be one of the following values:

ON turns borders on. ON generates the same line as MEDIUM.

Note: The MEDIUM line setting ensures consistency with lines created with GRID attributes.

OFF turns borders off. OFF is the default value.

LIGHT specifies a thin line.

MEDIUM identifies a medium line. **ON** sets the line to **MEDIUM**.

HEAVY identifies a thick line.

width specifies the line width in points, where 72 pts=1 inch.

Tip: Line width specified in points is displayed differently in HTML and PDF output. For uniform appearance, regardless of display format, use **LIGHT**, **MEDIUM**, or **HEAVY**.

position

Specifies which border line to format. Valid values are: **TOP**, **BOTTOM**, **LEFT**, **RIGHT**.

You can specify a position qualifier for any of the **BORDER** attributes. This enables you to format line width, line style, and line color individually, for any side of the border.

line_style

Sets the style of the border line. **FOCUS** StyleSheets support all of the standard Cascading Style Sheet line styles. Several 3-dimensional styles are available only in HTML, as noted by asterisks. Valid values are:

Style	Description
NONE	No border is drawn.
SOLID	Solid line.
DOTTED	Dotted line.
DASHED	Dashed line.
DOUBLE	Double line.
GROOVE*	3D groove.
RIDGE*	3D ridge.
INSET*	3D inset.
OUTSET*	3D outset.

color

Is one of the preset color values. The default value is **BLACK**.

If the display or output device does not support colors, it substitutes shades of gray. For a complete list of available color values, see [Color Values in a Report](#) on page 522.

RGB

Specifies the font color using a mixture of red, green, and blue.

(*r g b*)

Is the desired intensity of red, green, and blue, respectively. The values are on a scale of 0 to 255, where 0 is the least intense and 255 is the most intense. Using the three color components in equal intensities results in shades of gray.

Example: Inserting and Formatting a Border

This request generates an HTML report with a heavy red dotted line around the entire report heading.

```
TABLE FILE GGSales
SUM BUDUNITS UNITS BUDDOLLARS DOLLARS
BY CATEGORY
ON TABLE SUBHEAD
"</1 Sales Report"
"***CONFIDENTIAL***"
"December 2002 </1"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET HTMLCSS ON
ON TABLE SET STYLESHEET *
TYPE=REPORT, GRID=OFF, $
TYPE=TABLEHEADING, STYLE=BOLD, JUSTIFY=CENTER, BORDER=HEAVY,
BORDER-COLOR=RED, BORDER-STYLE=DOTTED, $
ENDSTYLE
END
```

The output is:

<p>Sales Report ***CONFIDENTIAL*** December 2002</p>				
---	--	--	--	--

<u>Category</u>	<u>Budget Units</u>	<u>Unit Sales</u>	<u>Budget Dollars</u>	<u>Dollar Sales</u>
Coffee	1385923	1376266	17293886	17231455
Food	1377564	1384845	17267160	17229333
Gifts	931007	927880	11659732	11695502

Tip: You can use the same BORDER syntax to generate this output in a PDF or PS report.

Example: Applying Grid Lines to All Cells of an HTML Report

This request uses GRID=FILL to apply grid lines to all cells, including those underneath the sort field CATEGORY. With GRID=ON, the cells underneath the sort field value do not have grid lines until the sort field value changes.

```
TABLE FILE GGSALES
SUM UNITS DOLLARS
BY CATEGORY BY PRODUCT
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=FILL, $
ENDSTYLE
END
```

All cells have grid lines:

Category	Product	Unit Sales	Dollar Sales
Coffee	Capuccino	189217	2381590
	Espresso	308986	3906243
	Latte	878063	10943622
Food	Biscotti	421377	5263317
	Croissant	630054	7749902
	Scone	333414	4216114
Gifts	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829

Syntax: **How to Add and Adjust Grid Lines (PDF or PS)**

This syntax applies to a PDF or PS report.

```
TYPE=type, {HGRID|VGRID}={ON|OFF|HEAVY}, $
```

where:

type

Identifies the report component to which grid lines are applied.

HGRID

Specifies horizontal grid lines.

VGRID

Specifies vertical grid lines.

ON

Applies light grid lines.

OFF

Suppresses grid lines. This is the default.

HEAVY

Applies heavy grid lines.

Example: **Applying Grid Lines to Report Data (PDF)**

This request applies light, horizontal grid lines to report data.

```
TABLE FILE GGDEMOG
HEADING
"State Statistics"
" "
SUM HH AS 'Number of,Households' AVGHHSZ98 AS 'Avg.,Size'
MEDHHI98 AS 'Avg.,Income'
BY ST
WHERE ST EQ 'CA' OR 'FL' OR 'NY'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=DATA, HGRID=ON, $
END
```

In the PDF report, the lines make it easier to distinguish the data by state:

State Statistics

<u>State</u>	<u>Number of Households</u>	<u>Avg. Size</u>	<u>Avg. Income</u>
CA	11478067	3	44925
FL	5968392	2	34264
NY	6799434	3	42023

Adding an Image to a Report

How to:

Add an Image to an HTML Report

Add a Background Image

Add an Image to a PDF, PS, or HTML Report With Internal Cascading Style Sheet

Reference:

Image Attributes

Specifying a URL

With a StyleSheet you can add and position an image in a report. An image, such as a logo, gives corporate identity to a report, or provides visual appeal. You can add more than one image by creating multiple declarations.

You can also add an image as background to a report. A background image is tiled or repeated, covering the entire area on which the report appears. An image attached to an entire report, or an image in a heading or footing, can appear with a background image.

Images must exist in a file format your browser supports, such as GIF (Graphic Interchange Format) or JPEG (Joint Photographic Experts Group). PDF and PS reports only support GIF files.

In an HTML report, the Web browser locates and displays the image, so it must be in a location that the browser can find. If the file is not on the search path, supply the full path name.

In a PDF or PS report, FOCUS reads the image and places it in the PDF or PS output file. Thus, the GIF file must reside on the CMS search path in z/VM or in a data set allocated to ddname GIF in z/OS.

Reference: Image Attributes

Attribute	Description
IMAGE	Adds an image.
IMAGEALIGN	Positions an image. This applies only to HTML reports.
POSITION	Positions an image.
IMAGEBREAK	Controls generation of a line break after an image. This applies only to HTML reports without internal Cascading Style Sheets.
SIZE	Sizes an image.
ALT	Supplies a description of an image for compliance with Section accessibility (Workforce Investment Act of 1998). ALT only applies to HTML reports.
BACKIMAGE	Adds a background image.

Syntax: How to Add an Image to an HTML Report

This syntax applies to an HTML report. For details on adding an image to a PDF, PS, or an HTML report with internal CSS, see *How to Add an Image to an HTML Report* on page 774.

```
TYPE={REPORT|heading}, IMAGE={url|(column)} [,IMAGEALIGN=position]
    [,IMAGEBREAK={ON|OFF}] [,ALT='description'], $
```

where:

REPORT

Embeds an image in the body of a report. This value is the default. **Note:** The IMAGE=(column) option is not supported with TYPE=REPORT.

heading

Embeds an image in a heading or footing. Valid values are TABHEADING, TABFOOTING, HEADING, FOOTING SUBHEAD, and SUBFOOT.

url

Is the URL for the image file. The image must exist in a separate file in a format that your browser supports, such as GIF or JPEG. The file can be on your local Web server, or on any server or directory accessible from your network. For details, see [Specifying a URL](#) on page 776.

column

Is an alphanumeric field in a request (for example, a display field or a BY field) whose value is a URL that points to an image file. Specify a value using the COLUMN attribute described in [Identifying Report Components](#) on page 525. Enclose *column* in parentheses.

This option enables you to add different images to a heading or footing, depending on the value of the field.

position

Is the position of the image. Valid values are:

TOP where the top right corner of the image aligns with heading or footing text. If the image is attached to the entire report, it appears on top of the report.

MIDDLE where the image appears in the middle of the heading or footing text. If the image is attached to the entire report, it appears in the middle of the report.

BOTTOM where the bottom right corner of the image aligns with heading or footing text. If the image is attached to the entire report, it appears at the bottom of the report.

LEFT where the image appears to the left of heading or footing text. If the image is attached to the entire report, it appears to the left of the report.

RIGHT where the image appears to the right of heading or footing text. If the image is attached to the entire report, it appears to the right of the report.

IMAGEBREAK

Controls generation of a line break after the image. Valid values are:

ON which generates a line break after the image so that an element following it (such as report heading text) appears on the next line.

OFF which suppresses a line break after the image so that an element following it is on the same line. This value is the default.

description

Is a textual description of an image for compliance with Section 508 accessibility. Enclose the description in single quotation marks.

Reference: Specifying a URL

The following guidelines are the same for `IMAGE=url` and `IMAGE=(column)` syntax. In the latter case, they apply to a URL stored in a data source field.

Specify a URL by:

- ❑ Supplying an absolute or relative address that points to an image file, for example:

```
TYPE=TABHEADING, IMAGE=http://www.ibi.com/images/logo_wf3.gif,$
TYPE=TABHEADING, IMAGE=/ibi_html/ggdemo/gotham.gif,$
```

- ❑ Using the `SET BASEURL` parameter to establish a URL that is logically prefixed to all relative URLs in the request. With this feature, you can add an image by specifying just its file name in the `IMAGE` attribute. For example:

```
SET BASEURL=D:\ibi\apps\SESSION\
.
.
.
TYPE=REPORT, IMAGE=gotham.gif,$
```

The following conditions apply:

- ❑ A base URL must end with a slash (/) or backslash (\).
- ❑ An absolute URL (which begins with `http://`) overrides a base URL.
- ❑ A URL is case sensitive when referring to a UNIX server.
- ❑ If the name of the image file does not contain an extension, GIF is used.

Example: Adding an Image to an HTML Report Heading

This request adds the Gotham Grinds logo to a report heading. The logo is in a separate image file identified by a relative URL in the `IMAGE` attribute.

```
TABLE FILE GGORDER
ON TABLE SUBHEAD
"PRODUCTS ORDERED ON 08/01/96"
SUM QUANTITY AS 'Ordered Units' BY PRODUCT
WHERE PRODUCT EQ 'Coffee Grinder' OR 'Coffee Pot'
WHERE ORDER_DATE EQ '08/01/96'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=TABHEADING, IMAGE=C:\IMAGES\GOTHAM.GIF, IMAGEBREAK=ON, $
ENDSTYLE
END
```

IMAGEBREAK, set to ON, generates a line break between the logo and the heading text:



PRODUCTS ORDERED ON 08/01/96

Product	Ordered Units
Coffee Grinder	2493
Coffee Pot	3100

Example: Using a File Name in a Data Source Field in an HTML Report

The following illustrates how to embed an image in a SUBHEAD, and use a different image for each value of the BY field on which the SUBHEAD occurs.

```
SET BASEURL=c:\images\  
  
DEFINE FILE CAR  
FLAG/A12=  
DECODE COUNTRY ( 'ENGLAND' 'uk' 'ITALY' 'italy'  
  'FRANCE' 'france' 'JAPAN' 'japan' );  
END
```

```
TABLE FILE CAR
PRINT FLAG NOPRINT AND MODEL AS '' BY COUNTRY NOPRINT AS '' BY CAR AS ''
WHERE COUNTRY EQ 'ENGLAND' OR 'FRANCE' OR 'ITALY' OR 'JAPAN'
ON COUNTRY SUBHEAD
"                               <+0>Cars produced in <ST.COUNTRY"
HEADING CENTER
"Car Manufacturer Report"
" "
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=SUBHEAD, IMAGE=(FLAG), IMAGEALIGN=TOP, $
TYPE=REPORT, GRID=OFF, $
TYPE=HEADING, SIZE=12, STYLE=BOLD, $
TYPE=SUBHEAD, STYLE=BOLD, $
ENDSTYLE
END
```

The output is:

Car Manufacturer Report

Cars produced in ENGLAND

JAGUAR	V12XKE AUTO
	XJ12L AUTO
JENSEN	INTERCEPTOR III
TRIUMPH	TR7

Cars produced in FRANCE

PEUGEOT	504 4 DOOR
---------	------------

Cars produced in ITALY

ALFA ROMEO	2000 4 DOOR BERLINA
	2000 GT VELOCE
	2000 SPIDER VELOCE
MASERATI	DOBA 2 DOOR

Cars produced in JAPAN

DATSUN	B210 2 DOOR AUTO
TOYOTA	COROLLA 4 DOOR LEX AUTO

Example: Supplying an Image Description Using the ALT Attribute

The following illustrates how to use the ALT attribute. The ALT attribute supplies a description of an image that screen readers can interpret to comply with Section 508 accessibility (Workforce Investment Act of 1998).

```
SET BASEURL=C:\images\  
TABLE FILE GGSALES  
SUM UNITS BY PRODUCT  
ON TABLE SUBHEAD  
"Report on Units Sold"  
ON TABLE SET PAGE-NUM OFF  
ON TABLE HOLD FORMAT HTML  
ON TABLE SET STYLE *  
TYPE=TABHEADING, IMAGE=gglogo, IMAGEBREAK=ON, POSITION=(.25 .25),  
    SIZE=(.5 .5), ALT='Gotham Grinds Logo Image', $  
GRID=OFF, $  
ENDSTYLE  
END
```

The output is:



Report on Units Sold

<u>Product</u>	<u>Unit Sales</u>
Biscotti	421377
Capuccino	189217
Coffee Grinder	186534
Coffee Pot	190695
Croissant	630054
Espresso	308986
Latte	878063
Mug	360570
Scone	333414
Thermos	190081

Syntax: **How to Add a Background Image**

This syntax applies to an HTML report.

```
[TYPE=REPORT,] BACKIMAGE=url, $
```

where:

```
TYPE=REPORT
```

Applies the image to the entire report. Not required, as it is the default.

```
url
```

Is the URL of a GIF or JPEG file. Specify a file on your local Web server, or on a server accessible from your network.

The URL can be an absolute or relative address. See [Image Attributes](#) on page 774.

When specifying a GIF file, you can omit the file extension.

Example: **Adding a Background Image**

This request adds a background image to a report. The image file CALM_BKG.GIF resides in the relative address shown.

```
TABLE FILE GGSales  
SUM UNITS DOLLARS  
BY CATEGORY BY PRODUCT  
ON TABLE SET PAGE-NUM OFF  
ON TABLE HOLD FORMAT HTML  
ON TABLE SET STYLE *  
TYPE=REPORT, STYLE=BOLD, GRID=OFF, $  
TYPE=REPORT, BACKIMAGE=C:\IMAGES\CALM_BKG.GIF, $  
ENDSTYLE  
END
```

The background is tiled across the report area:

<u>Category</u>	<u>Product</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
Coffee	Capuccino	189217	2381590
	Espresso	308986	3906243
	Latte	878063	10943622
Food	Biscotti	421377	5263317
	Croissant	630054	7749902
	Scone	333414	4216114
Gifts	Coffee Grinder	186534	2337567
	Coffee Pot	190695	2449585
	Mug	360570	4522521
	Thermos	190081	2385829

Syntax: **How to Add an Image to a PDF, PS, or HTML Report With Internal Cascading Style Sheet**

This syntax applies to a PDF, PS, or HTML report with internal Cascading Style Sheet.

```
TYPE={REPORT|heading}, IMAGE={url|file|(column)}
[,POSITION=( [+|-]x [+|-]y )] [,SIZE=(w h)] , $
```

where:

REPORT

Embeds an image in the body of a report. The image appears in the background of the report. This value is the default.

heading

Embeds an image in a heading or footing. Valid values are TABHEADING, TABFOOTING, FOOTING, HEADING, SUBHEAD, and SUBFOOT.

Provide sufficient blank space in the heading or footing so that the image does not overlap the heading or footing text. Also, you may want to place heading or footing text to the right of the image using spot markers or the POSITION attribute in the StyleSheet.

url

HTML report with internal Cascading Style Sheet:

Is the absolute or relative address for the image file. The image must exist in a separate file in a format that your browser supports, such as GIF or JPEG. The file can be on your local Web server, or on any server accessible from your network. For details, see [Specifying a URL](#) on page 776.

file

PDF or PS report:

Is the name of the image file. On VM, the file must be on the CMS search path. On z/OS, the file must reside in the PDS allocated to DDNAME GIF (with DCB attributes RECFM=FB,LRECL=1024). To transfer a GIF image to the Mainframe, use FTP in BINARY mode.

When specifying a GIF file, you can omit the file extension.

column

Is an alphanumeric field in the data source that contains the name of an image file. Use the COLUMN attribute described in [Identifying an Entire Report, Column, or Row](#) on page 527. Enclose *column* in parentheses.

The field containing the file name must be a display field or BY field referenced in the request.

Note that the value of the field is interpreted exactly as if it were typed as the URL of the image in the StyleSheet. If you omit the suffix, '.GIF' is supplied by default. SET BASEURL can be useful for supplying the base URL of the images; if you do this, the value of the field doesn't have to include the complete URL.

This syntax is useful, for example, if you want to embed an image in a SUBHEAD, and you want a different image for each value of the BY field on which the SUBHEAD occurs.

POSITION

Is the starting position of the image.

+|-

Measures the horizontal or vertical distance from the upper left corner of the report component in which the image is embedded.

x

Is the horizontal starting position of the image from the upper left corner of the physical report page, expressed in the unit of measurement specified by the UNITS parameter.

Enclose the x and y values in parentheses; do not include a comma between them.

y

Is the vertical starting position of the image from the upper left corner of the physical report page, expressed in the unit of measurement specified by the UNITS parameter.

SIZE

Is the size of the image. By default, an image is added at its original size.

w

Is the width of the image, expressed in the unit of measurement specified by the UNITS parameter.

Enclose the *w* and *h* values in parentheses; do not include a comma between them.

h

Is the height of the image, expressed in the unit of measurement specified by the UNITS parameter.

Example: Adding a GIF Image to an HTML Report With Internal Cascading Style Sheet

The TYPE attribute adds the image to the report heading. POSITION places the image .35 inch horizontally and .25 inch vertically from the upper left corner of the report page. The image is one inch wide and one inch high as specified by SIZE.

```

SET HTMLCSS = ON
TABLE FILE GGSALES
SUM UNITS BY PRODUCT
ON TABLE SUBHEAD
"REPORT ON UNITS SOLD"
" "
" "
" "
" "
" "
" "
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=TABLEHEADING, IMAGE=C:\IMAGES\GOTHAM.GIF,
POSITION=(.35 .25), SIZE=(1 1), $
ENDSTYLE
END

```

The company logo is positioned and sized in the report heading:

REPORT ON UNITS SOLD



<u>Product</u>	<u>Unit Sales</u>
Biscotti	421377
Capuccino	189217
Coffee Grinder	186534
Coffee Pot	190695
Croissant	630054
Espresso	308986
Latte	878063
Mug	360570
Scone	333414
Thermos	190081

Example: Adding a GIF Image to a PDF Report

The image file for this example is GOTHAM.GIF. The POSITION attribute places the image one-quarter inch horizontally and one-quarter vertically from the upper left corner of the report page. The image is one-half inch wide and one-half inch high as specified by SIZE.

```
TABLE FILE GGSALES
SUM UNITS BY PRODUCT
ON TABLE SUBHEAD
"Report on Units Sold"
" "
" "
" "
" "
" "
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=TABHEADING, IMAGE=GOTHAM.GIF, POSITION=(.25 .25), SIZE=(.5 .5), $
ENDSTYLE
END
```

The output is:

Report on Units Sold



<u>Product</u>	<u>Unit Sales</u>
Biscotti	421377
Capuccino	189217
Coffee Grinder	186534
Coffee Pot	190695
Croissant	630054
Espresso	308986
Latte	878063
Mug	360570
Scone	333414
Thermos	190081

Linking in a Report

In this section:

- Linking to a URL
- Linking to a JavaScript Function
- Linking With Conditions
- Linking From a Graphic Image
- Specifying a Base URL
- Specifying a Target Frame
- Linking Report Pages

You can use StyleSheet declarations to define links from any report component. You can create links from report data (including headings and footings) as well as graphic images (such as a company logo or product image).

Linking to a URL

How to:

Link to a URL

You can define a link from any report component to any URL. Once you have defined a link, you can select the report component to access the URL.

Syntax: **How to Link to a URL**

```
TYPE=type, [subtype], URL=url[(parameters ...)], [TARGET=frame, ]$
```

where:

type

Identifies the report component that you select in the Web browser to execute the link. The TYPE attribute and its value must appear at the beginning of the declaration.

subtype

Are any additional attributes, such as COLUMN, LINE, or ITEM, that are needed to identify the report component that you are formatting.

url

Identifies any valid URL, including a URL that specifies a CGI program, or the name of a report column enclosed in parentheses whose value is a valid URL to which the link will jump.

Note:

- ❑ The maximum length of a URL=*url* argument, including any associated variable=object parameters, is 2400 characters. The URL argument can span more than one line.
- ❑ If the URL refers to a CGI program that takes parameters, the URL must end with a question mark (?).

parameters

Values that are passed to the URL.

frame

Identifies the target frame in the Web page in which the output from the link is displayed. For details, see [Specifying a Target Frame](#) on page 794.

Example: Linking to a URL

The following example illustrates how to link to a URL from a report. The heading *Click here to access the IB homepage* is linked to the URL <http://www.ibi.com>. The relevant StyleSheet declarations are highlighted in the request.

```
TABLE FILE GGSALES
ON TABLE SET PAGE-NUM OFF
SUM UNITS AND DOLLARS
BY CATEGORY BY REGION
HEADING
"Regional Sales Report"
"Click here to access the IB homepage."
" "
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=HEADING, LINE=2, OBJECT=TEXT, ITEM=1,
URL=http://www.ibi.com, $
ENDSTYLE
END
```

The output is:

Regional Sales Report
[Click here to access the IB homepage.](http://www.ibi.com)

<u>Category</u>	<u>Region</u>	<u>Unit Sales</u>	<u>Dollar Sales</u>
Coffee	Midwest	332777	4178513
	Northeast	335778	4164017
	Southeast	350948	4415408
	West	356763	4473517
Food	Midwest	341414	4338271
	Northeast	353368	4379994
	Southeast	349829	4308731
	West	340234	4202337
Gifts	Midwest	230854	2883881
	Northeast	227529	2848289
	Southeast	234455	2986240
	West	235042	2977092

When you click the link, the site appears in your browser.

Linking to a JavaScript Function

How to:

Link to a JavaScript Function

You can use a StyleSheet to define a link to a JavaScript function from any report component. Once you have defined the link, you can select the report component to execute the JavaScript function.

You can specify optional parameters that allow values to be passed to the JavaScript function. The function will use the passed value to dynamically determine the results that are returned to the browser.

Note:

- ❑ JavaScript functions can, in turn, call other JavaScript functions.
- ❑ You cannot specify a target frame if you are executing a JavaScript function. However, the JavaScript function itself can specify a target frame for its results.

Syntax: **How to Link to a JavaScript Function**

`TYPE=type, [subtype], JAVASCRIPT=function[(parameters ...)], $`

where:

type

Identifies the report component that you select in the Web browser to execute the link. The TYPE attribute and its value must appear at the beginning of the declaration.

subtype

Are any additional attributes, such as COLUMN, LINE, or ITEM, that are needed to identify the report component that you are formatting.

function

Identifies the JavaScript function to run when you select the report component.

The maximum length of a JAVASCRIPT=function argument, including any associated parameters, is 2400 characters and can span more than one line. If you split a single argument across a line, you must use the \ character at the end of the first line, as continuation syntax. If you split an argument at a point where a space is required as a delimiter, the space must be before the \ character or be the first character on the next line. The \ character does not act as the delimiter.

parameters

Values that are passed to the JavaScript function.

Linking With Conditions

How to:

Link With Conditions

You can create conditions when linking from a report. For example, you may be interested in displaying only current salaries for a particular department. You can accomplish this by creating a WHEN condition.

Syntax: **How to Link With Conditions**

To specify a conditional link to a URL use:

```
TYPE=type, [subtype], URL=url[(parameters...)],
  WHEN=expression,[TARGET=frame,] $
```

To specify a conditional link to a JavaScript function use

```
TYPE=type, [subtype], JAVASCRIPT=function[(parameters...)],
  WHEN=expression,[TARGET=frame,] $
```

where:

type

Identifies the report component that you select in the Web browser to execute the link. The TYPE attribute and its value must appear at the beginning of the declaration.

subtype

Are any additional attributes, such as COLUMN, LINE, or ITEM, that are needed to identify the report component that you are formatting.

url

Identifies any valid URL, or the name of a report column enclosed in parentheses whose value is a valid URL.

function

Identifies the JavaScript function to run when you select the report component.

parameters

Values that are passed to the URL.

expression

Is any Boolean expression that would be valid on the right side of a COMPUTE expression.

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 within single quotation marks.

frame

Identifies the target frame in the Web page in which the output from the link is displayed. For details, see [Specifying a Target Frame](#) on page 794.

Example: Linking With Conditions

Assume you want to link only the MIS value of the DEPARTMENT field to the URL. To do this we include the phrase WHEN=DEPARTMENT EQ 'MIS' in the StyleSheet declaration. The relevant declarations are highlighted in the requests.

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AS 'Total,Current,Salaries'
BY DEPARTMENT AS 'Department'
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=DATA, COLUMN=N1, URL=http://www.informationbuilders.com,
WHEN=DEPARTMENT EQ 'MIS', $
ENDSTYLE
END
```

In the following output, note that only the MIS department is linked:

	Total
	Current
<u>Department</u>	<u>Salaries</u>
MIS	\$108,002.00
PRODUCTION	\$114,282.00

Linking From a Graphic Image

How to:

Specify Links From a Graphic Image

You can link from an image in an HTML report. The image can be attached to the entire report or to the report heading or footing (this includes table headings/table footings, and sub-headings/sub-footings).

The syntax for linking from a graphic image is the same as when linking from a report component. The only difference is that you add `IMAGE=image` to the StyleSheet declaration.

Note: You can only link from an image when you are using HTML format.

Syntax: How to Specify Links From a Graphic Image

```
TYPE=type, [subtype], IMAGE=image, URL=url
  [(parameters ...)],[TARGET=frame,] $
```

where:

type

Identifies the report component that the user selects to execute the link. The TYPE attribute and its value must appear at the beginning of the declaration. You can specify the following types of components:

`TABHEADING` or `TABFOOTING` enables you to link from a graphical image that is attached to a report heading or footing.

`HEADING` or `FOOTING` enables you to link from a graphical image that is attached to a page heading or footing.

`SUBHEAD` or `SUBFOOT` enables you to link from a graphical image that is attached to a sub heading or sub footing.

subtype

Are any additional attributes, such as `COLUMN`, `LINE`, or `ITEM`, that are needed to identify the report component that you are formatting.

image

Specifies the file name of a graphical image file. The image must exist as a separate graphic file in a format that your browser supports. Most browsers support GIF and JPEG file types.

You can specify a local image file, or identify an image elsewhere on the network using a URL. URLs can be absolute, such as `http://www.ibi.com/graphic.gif`, or relative aliases that can be identified to the Web server, such as `/aproot/ibincen/graphic.gif`.

Alternatively, you can specify an alphanumeric field in the report (either a BY sort field or a display field) whose value corresponds to the name of the image file.

url

Identifies any valid URL, or the name of a report column enclosed in parentheses whose value is a valid URL.

parameters

Are values that are passed to the URL. You can pass one or more parameters. The entire string of parameters must be enclosed in parentheses, and separated from each other by a blank space.

frame

Identifies the target frame in the Web page in which the output from the link is displayed. For details, see [Specifying a Target Frame](#) on page 794.

Example: Specifying a Link From an Image

The following example illustrates how to link a URL from an image. The relevant StyleSheet declarations are highlighted in the request.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME BY EMP_ID
HEADING
"List Of Employees By Employee ID"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=HEADING, STYLE=BOLD, $
TYPE=REPORT, GRID=OFF, $
TYPE=REPORT, IMAGE=C:\IMAGES\LEFTLOGO.GIF,
URL=HTTP://INFORMATIONBUILDERS.COM, $
ENDSTYLE
END
```

The output is:



List Of Employees By Employee ID

<u>EMP ID</u>	<u>LAST NAME</u>
071382660	STEVENS
112847612	SMITH
117593129	JONES
119265415	SMITH
119329144	BANNING
123764317	IRVING
126724188	ROMANS
219984371	MCCOY
326179357	BLACKWOOD
451123478	MCKNIGHT
543729165	GREENSPAN
818692173	CROSS

When you click the graphic, the Information Builders Web site opens.

Specifying a Base URL

How to:

Specify a Base URL

If you want to create a link but do not know the full, physical URLs, you can specify a default location where the browser searches for relative URLs.

To specify a default URL location, use the SET BASEURL command. Using SET BASEURL puts `<BASE HREF="url">` into the HTML file that FOCUS generates. When a report is run, the specified directory is searched for the files that are called by the generated Web page.

Syntax: **How to Specify a Base URL**

```
SET BASEURL=url
```

where:

```
url
```

Is the fully-qualified directory in which additional files reside. If it points to a Web server, the URL must begin with `http://`. The URL MUST end with a closing delimiter (`/` or `\`).

Example: **Specifying a Base URL**

The following illustrates how to specify a base URL:

```
SET BASEURL=http://www.informationbuilders.com/ibi\_html/newcentcorp/images/
```

If you are including a graphic image in your report that is stored in the specified base URL, you can add the following declaration to your StyleSheet instead of typing the entire URL:

```
TYPE=HEADING, IMAGE=leftlogo.gif, ..., $
```

Note: If the URL is at a remote website, it may take longer to retrieve. Whenever possible, store graphic image files on your system.

Specifying a Target Frame

How to:

- Specify a Target Frame
- Specify a Default Target Frame

You can use frames to subdivide application HTML pages into separate scrollable sections. Frames enable users to explore various information items on a page by scrolling through a section, instead of linking to a separate page. When defining a link from a report component, you can specify that the results of the link be displayed in a target frame on a Web page.

There are two ways to specify a target frame. You can specify:

- ❑ A target frame in a StyleSheet declaration using the TARGET attribute. You can use StyleSheets to specify that links from a report or graph are displayed in a target frame on the Web page displaying the report or graph. However, using StyleSheets to specify target frames adds extra HTML syntax to every HREF that is generated.
- ❑ A default target frame with a SET command. SET TARGETFRAME puts the HTML code `<BASE TARGET="framename">` into the header of the HTML file that FOCUS opens. All links from the report are directed to the specified frame, unless overridden by the TARGET attribute in the StyleSheet.

To use the TARGET attribute or the SET TARGETFRAME command, you must create multiple frames on the Web page.

Syntax: How to Specify a Target Frame

To specify a target frame for a URL, use

```
TYPE=type, [subtype], URL=url[(parameters ...)], [TARGET=frame,] $
```

where:

type

Identifies the report component that the user selects in the Web browser to execute the link. The TYPE attribute and its value must appear at the beginning of the declaration.

subtype

Are any additional attributes, such as COLUMN, LINE, or ITEM, that are needed to identify the report component that you are formatting.

url

Identifies any valid URL, or the name of a report column enclosed in parentheses whose value is a valid URL to which the link will jump. For details about linking to an URL, see [Linking to a URL](#) on page 786.

parameters

Are values being passed to the URL. You can pass one or more parameters. The entire string of values must be enclosed in parentheses, and separated from each other by a blank space.

frame

Identifies the target frame in the Web page in which the output from the link is displayed.

If the name of the target frame contains embedded spaces, the name will be correctly interpreted without enclosing the name in quotation marks. For example:

```
TYPE=DATA, COLUMN=N1,  
FOCEXEC=MYREPORT, TARGET=MY FRAME, $
```

The name of the target frame is correctly interpreted to be MY FRAME.

You can also use the following standard HTML frame names: _BLANK, _SELF, _PARENT, _TOP.

Syntax: **How to Specify a Default Target Frame**

```
SET TARGETFRAME=frame
```

where:

frame

Identifies the target frame in the Web page in which the output from the link is displayed.

Example: **Specifying a Target Frame**

The following illustrates how to specify a default target frame:

```
SET TARGETFRAME=_SELF
```

The following illustrates how to specify a target frame in a request. The relevant StyleSheet declaration is highlighted in the request.

```
TABLE FILE EMPLOYEE  
PRINT CURR_SAL  
BY DEPARTMENT  
ON TABLE HOLD FORMAT HTML  
ON TABLE SET STYLE *  
TYPE=DATA, COLUMN=N1, URL=http:\\www.informationbuilders.com,  
    TARGET=_SELF, $  
ENDSTYLE  
END
```

Linking Report Pages

How to:

Link Report Pages

With a StyleSheet, you can insert one or more navigational hyperlinks in a multi-page HTML report. This feature makes it easy for you to link consecutive report pages together without creating individual hyperlinks for each page.

You can define any report component as a hyperlink.

Syntax: How to Link Report Pages

Use the following syntax in an HTML report

```
URL=#_destination, $
```

where:

destination

Is one of the following:

#_next goes to the top of the next report page.

#_previous goes to the top of the previous report page.

#_top goes to the top of the current report page.

#_start goes to the first page of the report.

#_end goes to the last page of the report.

Example: Linking Report Pages Through Images in a Heading

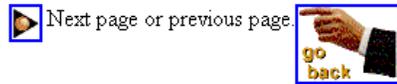
This request displays two images in the page heading of a long report. It creates a link between BULLET.GIF and the next page of the report, and GOBACK.GIF and the previous page of the report.

```
TABLE FILE GGORDER
ON TABLE SUBHEAD
"COFFEE GRINDER SALES BY STORE"
" "
HEADING
"Next page or previous page."
PRINT QUANTITY AS 'Ordered Units' BY STORE_CODE BY PRODUCT NOPRINT
BY ORDER_NUMBER
WHERE PRODUCT EQ 'Coffee Grinder'
ON STORE_CODE PAGE-BREAK
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=TABHEADING, STYLE=BOLD,$
TYPE=HEADING, IMAGE=C:\IMAGES\BULLET, URL=#_next, IMAGEALIGN=LEFT,$
TYPE=HEADING, IMAGE=C:\IMAGES\GOBACK, URL=#_previous, IMAGEALIGN=RIGHT,$
ENDSTYLE
END
```

The images display in each page heading.

PAGE 1

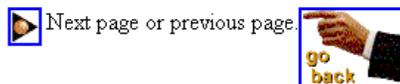
COFFEE GRINDER SALES BY STORE



<u>Store Code</u>	<u>Order Number</u>	<u>Ordered Units</u>
R1019	9	265
	189	148
	369	382
	549	325
	729	381
	909	41
	1086	295
	1266	221

Click the image on the left of page 1 to display page 2:

PAGE 2



<u>Store Code</u>	<u>Order Number</u>	<u>Ordered Units</u>
R1020	24	133
	204	380
	384	279
	564	357
	744	189
	924	90
	1101	248
	1281	221
	1461	262
	1641	83
	1821	184

Click the "go back" image on page 2 to redisplay page 1.

Example: Linking Pages Through Page Number and Heading Elements

This request creates hyperlinks from the page number to the next page in the report, and from the text of the page heading, which appears at the top of every report page, back to the previous page or to the first page.

```
TABLE FILE GGORDER
ON TABLE SUBHEAD
"COFFEE GRINDER SALES BY STORE"
" "
HEADING
"return to previous page"
"return to beginning"
PRINT QUANTITY AS 'Ordered Units' BY STORE_CODE BY PRODUCT NOPRINT
BY ORDER_NUMBER
WHERE PRODUCT EQ 'Coffee Grinder'
ON STORE_CODE PAGE-BREAK
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
TYPE=TABHEADING, STYLE=BOLD,$
TYPE=PAGENUM, URL=#_next, $
TYPE=HEADING, LINE=1, URL=#_previous, $
TYPE=HEADING, LINE=2, URL=#_start, $
ENDSTYLE
END
```

The first page is:

[PAGE 1](#)

COFFEE GRINDER SALES BY STORE

[return to previous page](#)

[return to beginning](#)

<u>Store Code</u>	<u>Order Number</u>	<u>Ordered Units</u>
R1019	9	265
	189	148
	369	382
	549	325
	729	381
	909	41
	1086	295
	1266	221
	1446	122

Click the page number three times to move to PAGE 4:

[PAGE 4](#)

[return to previous page](#)

[return to beginning](#)

<u>Store Code</u>	<u>Order Number</u>	<u>Ordered Units</u>
R1041	54	292
	234	165
	414	229
	594	208
	774	294
	953	276
	1131	137
	1311	278
	1491	289

Click *previous page* to return to PAGE 3. Click *return to beginning* to go directly to PAGE 1.

Working With Mailing Labels and Multi-Pane Pages

How to:

Set Up a Report to Print Mailing Labels

Print Mailing Labels or a Multi-Pane Report

Reference:

Attributes for Mailing Labels and Multi-Pane Printing

You can print sheets of mailing labels by dividing each page into a matrix of sub-pages, each corresponding to a single label. Each page break in the report positions the printer at the top of the next label.

Multi-pane printing places a whole report on a single printed page. You can create columns or rows so that when text overflows on one page, it appears in the next column or row on the same page rather than on the next page.

These features apply to a PDF or PS report.

Reference: [Attributes for Mailing Labels and Multi-Pane Printing](#)

In addition to the attributes in the table, you can use standard margin attributes (for example, LEFTMARGIN or TOPMARGIN) to position the entire sheet of labels at once, creating an identical margin for each sheet.

Attribute	Description	Applies to
PAGEMATRIX	Sets the number of columns and rows of labels on a page.	PDF PS
ELEMENT	Sets the width and height of each label, expressed in the unit of measurement specified by the UNITS parameter.	PDF PS
GUTTER	Sets the horizontal and vertical distance between each label, expressed in the unit of measurement specified by the UNITS parameter.	PDF PS
MATRIXORDER	Sets the order in which the labels are printed.	PDF PS

Attribute	Description	Applies to
LABELPROMPT	Sets the position of the first label on the mailing label sheet.	PDF PS

Procedure: How to Set Up a Report to Print Mailing Labels

1. Create the label as a page heading.
2. Sort the labels but use NOPRINT to suppress sort field display. Only the fields embedded in the page heading will print.
3. Insert a page break on a sort field to place each new field value on a separate label.
4. Suppress default page numbers and associated blank lines from the beginning of each page (SET PAGE-NUM=NOPAGE).

Syntax: How to Print Mailing Labels or a Multi-Pane Report

```
[TYPE=REPORT,] PAGEMATRIX=(cr), ELEMENT=(w h), [GUTTER=(x y),]
[MATRIXORDER={VERTICAL|HORIZONTAL},] [LABELPROMPT={OFF|ON},] $
```

where:

TYPE=REPORT

Applies the settings to the entire report. Not required, as it is the default.

c

Is the number of columns of labels across the page.

Enclose the values *c* and *r* in parentheses, and do not include a comma between them.

r

Is the number of rows of labels down the page.

w

Is the width of each label.

Enclose the values *w* and *h* in parentheses, and do not include a comma between them.

h

Is the height of each label.

GUTTER

Is the distance between each label.

`x`

Is the horizontal distance between each label.

Enclose the values `x` and `y` in parentheses, and do not include a comma between them.

`y`

Is the vertical distance between each label.

`MATRIXORDER`

Is the order in which the labels are printed.

`VERTICAL`

Prints the labels down the page.

`HORIZONTAL`

Prints the labels across the page.

`LABELPROMPT`

Is the position of the first label on the mailing label sheet.

`OFF`

Starts the report on the first label on the sheet. This value is the default.

`ON`

Prompts you at run time for the row and column number at which to start printing. All remaining labels follow consecutively. This feature allows partially used sheets of labels to be reused.

Example: Printing Mailing Labels

The following report prints on 8 1/2 x 11 sheets of address labels.

```
TABLE FILE EMPLOYEE
BY LAST_NAME NOPRINT BY FIRST_NAME NOPRINT
ON FIRST_NAME PAGE-BREAK
HEADING
"<FIRST_NAME <LAST_NAME "
"<ADDRESS_LN1 "
"<ADDRESS_LN2 "
"<ADDRESS_LN3 "
ON TABLE SET PAGE-NUM NOPAGE
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE LABEMP
END
```

The labels have the following dimensions, defined in the StyleSheet LABEMP:

```
UNITS=IN, PAGESIZE=LETTER, LEFTMARGIN=0.256, TOPMARGIN=0.5,  
PAGEMATRIX=(2 5), ELEMENT=(4 1), GUTTER=(0.188 0), $
```

The first page of labels prints as follows:

```
JOHN BANNING  
160 LOMBARDO AVE.  
APT 4C  
FREEPORT NY 11520
```

```
DIANE JONES  
235 MURRAY HIL PKWY  
RUTHERFORD NJ 07073
```

```
ROSEMARIE BLACKWOOD  
MRS. P. JONES  
3704 FARRAGUT RD.  
BROOKLYN NY 11210
```

```
JOHN MCCOY  
ASSOCIATED  
2 PENN PLAZA  
NEW YORK NY 10001
```

```
BARBARA CROSS  
APT 2G  
147-15 NORTHERN BLD  
FLUSHING NY 11354
```

```
ROGER MCKNIGHT  
APT 4D  
117 HARRISON AVE.  
ROSELAND NJ 07068
```

```
MARY GREENSPAN  
13 LINDEN AVE.  
JERSEY CITY NJ 07300
```

```
ANTHONY ROMANS  
271 PRESIDENT ST.  
FREEPORT NY 11520
```

```
JOAN IRVING  
APT 2J  
123 E 32 ST.  
NEW YORK NY 10001
```

```
MARY SMITH  
ASSOCIATED  
2 PENN PLAZA  
NEW YORK NY 10001
```

Example: Printing a Multi-Page Report

This request divides the first report page in two columns so that the second report page appears in the second column of the first page. A PAGE-BREAK creates a multi-page report for the purpose of this example.

```
TABLE FILE EMPLOYEE  
PRINT LAST_NAME AND CURR_SAL BY DEPARTMENT  
ON DEPARTMENT PAGE-BREAK  
HEADING  
"PAGE <TABPAGENO"  
ON TABLE HOLD FORMAT PDF  
ON TABLE SET STYLE *  
UNITS=IN, PAGESIZE=LETTER, PAGEMATRIX=(2 1), ELEMENT=(3.5 8.0),  
MATRIXORDER=VERTICAL, $  
TYPE=REPORT, SIZE=8, $  
END
```

The report prints as:

PAGE 1			PAGE 2		
<u>DEPARTMENT</u>	<u>LAST_NAME</u>	<u>CURR_SAL</u>	<u>DEPARTMENT</u>	<u>LAST_NAME</u>	<u>CURR_SAL</u>
MIS	SMITH	\$13,200.00	PRODUCTION	STEVENS	\$11,000.00
	JONES	\$18,480.00		SMITH	\$9,500.00
	MCCOY	\$18,480.00		BANNING	\$29,700.00
	BLACKWOOD	\$21,780.00		IRVING	\$26,862.00
	GREENSPAN	\$9,000.00		ROMANS	\$21,120.00
	CROSS	\$27,062.00		MCKNIGHT	\$16,100.00

15 Handling Records With Missing Field Values

Missing data is defined as data that is missing from a report because it is not relevant or 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 808.
- ❑ A field in a segment instance does not have a data value. See [Missing Field Values](#) on page 809.
- ❑ A parent segment instance does not have child instances (missing segment instances). See [Handling a Missing Segment Instance](#) on page 822.

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.

Topics:

- ❑ Irrelevant Report Data
- ❑ Missing Field Values
- ❑ Handling a Missing Segment Instance
- ❑ Setting the NODATA Character String

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 828).

Example: Irrelevant Report Data

The following request shows how the default NODATA character displays missing data in a report.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY LAST_NAME
BY FIRST_NAME
ACROSS DEPARTMENT
END
```

The output is:

LAST_NAME	FIRST_NAME	DEPARTMENT	
		MIS	PRODUCTION
BANNING	JOHN	.	\$29,700.00
BLACKWOOD	ROSEMARIE	\$21,780.00	.
CROSS	BARBARA	\$27,062.00	.
DAVIS	ELIZABETH	\$.00	.
GARDNER	DAVID	.	\$.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 the MIS 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

In this section:

MISSING Attribute in the Master File

MISSING Attribute in a DEFINE or COMPUTE Command

Testing for a Segment With a Missing Field Value

Preserving Missing Data Values in an Output File

Propagating Missing Values to Reformatted Fields in a Request

Missing values within segment instances occur when the instances exist, but some of the fields lack values.

When fields in instances lack values, numeric fields are assigned the value 0, and alphanumeric fields, the value 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 828.

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 yet 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 have 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), the value 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 or COMPUTE Command](#) on page 812.

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.) 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 only affect 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 or COMPUTE Command* on page 812).

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:

X>Returns COUNT	RETURNS COUNT	AVE X>Returns	AVE 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 814.

MISSING Attribute in a DEFINE or COMPUTE Command

How to:

Specify Missing Values in a DEFINE or COMPUTE Command

You can set the MISSING attribute ON in a DEFINE or COMPUTE command to enable a temporary field with missing values to be interpreted and represented correctly in reports.

An expression used to derive the values of the temporary field can contain real fields that have missing values. However, when used to derive the value of a temporary field, a data source field that is missing a value is evaluated as 0 or blank for computational purposes, 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 temporary fields, you can set the MISSING attribute ON for the virtual field in the DEFINE or COMPUTE 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 or COMPUTE Command](#) on page 812.

Syntax: How to Specify Missing Values in a DEFINE or COMPUTE Command

```
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 temporary 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 temporary field has a value (the missing values of the field are evaluated as 0 or blank in the calculation). If all of the fields in the expression are missing values, the temporary field is missing its value. SOME is the default value.

ALL

Indicates that if all the fields in the expression have values, the temporary field has a value. If at least one field in the expression has a missing value, the temporary 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 temporary 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:

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:

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
		E3	4	2	6	6
NEWARK	10/18	B10	1	1	2	2
	10/19	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

How to:

Test 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 set the ALL parameter to PASS to test for missing instances. See [Handling a Missing Segment Instance](#) on page 822.

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:

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:

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:

CITY	DATE	PROD_CODE	RETURNS
----	----	-----	-----
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. The first 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:

CITY	DATE	PROD_CODE	RETURNS
----	----	-----	-----
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

How to:

Distinguish Missing Data in an Extract File
Store Missing Data in HOLD Files

Reference:

Usage Notes for Holding Missing Values

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, by default the NODATA string can be stored in the output file. You can also use the SET HOLDMISS command to store the missing values rather than the NODATA character in an output file. For related information, see [Saving and Reusing Your Report Output](#) on page 421.

Syntax: **How to Distinguish Missing Data in an Extract File**

```
ON TABLE {HOLD|SAVE|SAVB} MISSING {ON|OFF}
```

where:

HOLD

Creates an extract file for use in subsequent reports. The default for MISSING is ON.

SAVE

Creates a text extract file for use in other programs. The default for MISSING is OFF.

SAVB

Creates a binary extract file for use in 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 following request specifies MISSING ON in the HOLD phrase:

```
TABLE FILE SALES  
SUM RETURNS AND HOLD FORMAT ALPHA MISSING ON  
BY CITY BY DATE BY PROD_CODE  
END
```

The MISSING=ON attribute for the RETURNS field is propagated to the HOLD Master File. In addition, the missing data symbols are propagated to the HOLD file for the missing field values:

```
FILENAME=HOLD      , SUFFIX=FIX      , $
  SEGMENT=HOLD, SEGTYPE=S3, $
    FIELDNAME=CITY, ALIAS=E01, USAGE=A15, ACTUAL=A15, $
    FIELDNAME=DATE, ALIAS=E02, USAGE=A4MD, ACTUAL=A04, $
    FIELDNAME=PROD_CODE, ALIAS=E03, USAGE=A3, ACTUAL=A03, $
    FIELDNAME=RETURNS, ALIAS=E04, USAGE=I3, ACTUAL=A03, $
      MISSING=ON, $
```

With MISSING OFF in the HOLD phrase, the MISSING=ON attribute is not propagated to the HOLD Master File and the missing data symbols are replaced with default values.

Syntax: How to Store Missing Data in HOLD Files

```
SET HOLDMISS={ON|OFF}
ON TABLE SET HOLDMISS {ON|OFF}
```

where:

ON

Allows you to store missing data in a HOLD file. When TABLE generates a default value for data not found, it generates missing values.

OFF

Does not allow you to store missing data in a HOLD file. OFF is the default value.

Reference: Usage Notes for Holding Missing Values

- ❑ Setting HOLDMISS ON adds the MISSING=ON attribute to every field in the extract file.
- ❑ Data is not found if:
 - ❑ ALL is set to ON.
 - ❑ The request is multi-path.
 - ❑ An ACROSS statement has been issued.

Example: Holding Missing Values Using HOLDMISS

```
SET HOLDMISS=ON
TABLE FILE MOVIES
  SUM WHOLESALPR
  BY CATEGORY ACROSS RATING
  ON TABLE HOLD AS HLDM
END
TABLE FILE HLDM
  PRINT *
END
```

The output is:

CATEGORY	WHOLESALPR	WHOLESALPR	WHOLESALPR	WHOLESALPR	WHOLESALPR
ACTION	.	.	20.98	.	34.48
CHILDREN	54.49	51.38	.	.	.
CLASSIC	40.99	160.80	.	.	.
COMEDY	.	.	46.70	30.00	13.75
DRAMA	10.00
FOREIGN	13.25	.	62.00	.	70.99
MUSICALS	15.00	.	13.99	9.99	13.99
MYSTERY	.	9.00	18.00	9.00	80.97
SCI/FI	.	.	.	35.99	43.53
TRAIN/EX	.	60.98	.	.	.

Propagating Missing Values to Reformatted Fields in a Request

How to:
Control Missing Values in Reformatted Fields

Reference:
Usage Notes for SET COMPMISS

When a field is reformatted in a request (for example, SUM field/format), an internal COMPUTE field is created to contain the reformatted field value and display on the report output. If the original field has a missing value, that missing value can be propagated to the internal field by setting the COMPMISS parameter ON. If the missing value is not propagated to the internal field, it displays a zero (if it is numeric) or a blank (if it is alphanumeric). If the missing value is propagated to the internal field, it displays the missing data symbol on the report output.

Syntax: How to Control Missing Values in Reformatted Fields

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

where:

ON

Propagates a missing value to a reformatted field.

OFF

Displays a blank or zero for a reformatted field. OFF is the default value.

Note: The COMPMISS parameter cannot be set in an ON TABLE command.

Example: Controlling Missing Values in Reformatted Fields

The following procedure prints the RETURNS field from the SALES data source for store 14Z. With COMPMISS OFF, the missing values display as zeros in the column for the reformatted field value. (**Note:** Before trying this example, you must make sure that the SALEMISS procedure, which adds missing values to the SALES data source, has been run.)

```
SET COMPMISS = OFF
TABLE FILE SALES
PRINT RETURNS RETURNS/D12.2 AS 'REFORMATTED,RETURNS'
BY STORE_CODE
WHERE STORE_CODE EQ '14Z'
END
```

The output is:

STORE_CODE	RETURNS	REFORMATTED RETURNS
-----	-----	-----
14Z	2	2.00
	2	2.00
	0	.00
	.	.00
	4	4.00
	0	.00
	3	3.00
	4	4.00
	.	.00
	4	4.00

With COMPMISS ON, the column for the reformatted version of RETURNS displays the missing data symbol when a value is missing:

```
SET COMPMISS = ON
TABLE FILE SALES
PRINT RETURNS RETURNS/D12.2 AS 'REFORMATTED,RETURNS'
BY STORE_CODE
WHERE STORE_CODE EQ '14Z'
END
```

The output is:

STORE_CODE	RETURNS	REFORMATTED RETURNS
14Z	2	2.00
	2	2.00
	0	.00
	.	.
	4	4.00
	0	.00
	3	3.00
	4	4.00
	.	.
	4	4.00

Reference: Usage Notes for SET COMPMISS

If you create a HOLD file with COMPMISS ON, the HOLD Master File for the reformatted field indicates MISSING = ON (as does the original field). With COMPMISS = OFF, the reformatted field does NOT have MISSING = ON in the generated Master File.

Handling a Missing Segment Instance

In this section:

- Including Missing Instances in Reports With the ALL. Prefix
- Including Missing Instances in Reports With the SET ALL Parameter
- Testing for Missing Instances in FOCUS Data Sources

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.

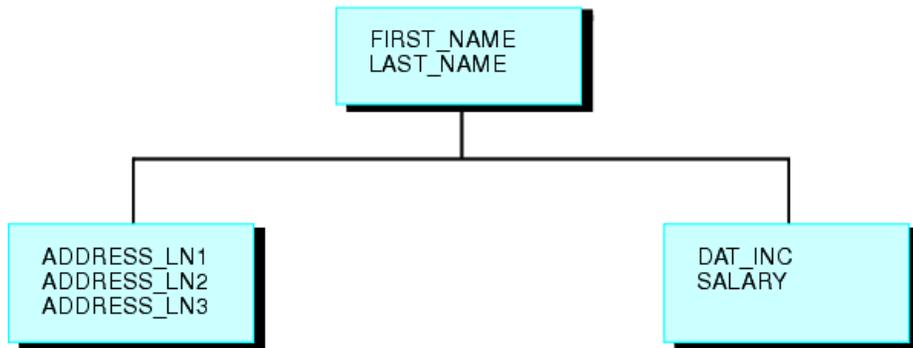
When you write a request from a data source that has missing segment instances, the missing instances affect the report. For example, if the request names fields in a segment and its descendants, the report omits parent segment instances that have no descendants. It makes no difference whether fields 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.

- ❑ The top segment contains employee names.
- ❑ The left segment contains addresses.

- The right segment contains the salary history of each employee: 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 parameter 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 823 and [Reporting Against Segments With Descendant Instances](#) on page 824.

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:

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:

LAST_NAME	FIRST_NAME	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 these instances, use the SET ALL=PASS command described in [Including Missing Instances in Reports With the SET ALL Parameter](#) on page 826.

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:

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 Parameter

How to:

Include a Parent Instance With Missing Descendants

You can include parent instances with missing descendants by issuing the SET ALL parameter 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, if a test on a missing segment fails, this 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

The following request displays all employees, regardless of whether they have taken a course or not since the ALL=PASS command is set.

If the ALL=ON command had been used, employees that had not taken courses would have been omitted because of the WHERE criteria.

```
JOIN EMPDATA.PIN IN EMPDATA TO ALL TRAINING.PIN IN TRAINING AS JOIN1
SET ALL = PASS
TABLE FILE EMPDATA
PRINT LASTNAME AND FIRSTNAME AND COURSECODE AND EXPENSES
BY PIN
WHERE EXPENSES GT 3000
END
```

The output is:

PIN	LASTNAME	FIRSTNAME	COURSECODE	EXPENSES
000000020	BELLA	MICHAEL	.	.
000000040	ADAMS	RUTH	EDP750	3,400.00
000000050	ADDAMS	PETER	UMI720	3,300.00
000000060	PATEL	DORINA	.	.
000000070	SANCHEZ	EVELYN	.	.
000000080	SO	PAMELA	BIT420	3,350.00
	SO	PAMELA	EDP690	3,200.00
000000090	PULASKI	MARIANNE	.	.
000000100	ANDERSON	TIM	NAMA930	3,100.00
000000130	CVEK	MARCUS	.	.
000000140	WHITE	VERONICA	BIT420	3,600.00
000000150	WHITE	KARL	UNI780	3,400.00
000000170	MORAN	WILLIAM	.	.
000000190	MEDINA	MARK	EDP690	3,150.00
000000220	LEWIS	CASSANDRA	.	.
000000230	NOZAWA	JIM	.	.
000000300	SOPENA	BEN	.	.
000000340	GOTLIEB	CHRIS	EDP750	3,450.00
	GOTLIEB	CHRIS	SSI670	3,300.00
000000350	FERNSTEIN	ERWIN	UMI720	3,350.00
000000380	ELLNER	DAVID	UNI780	3,350.00
000000390	GRAFF	ELAINE	.	.
000000400	LOPEZ	ANNE	.	.
000000410	CONTI	MARSHALL	EDP690	3,100.00

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 displays only the parent instances that are missing descendants.

Example: Testing for a MISSING Instance

The following request screens all salary instances, since no SALARY can both be 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:

LAST_NAME	FIRST_NAME
DAVIS	ELIZABETH
GARDNER	DAVID

Setting the NODATA Character String

How to:
Set the NODATA 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:

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

16 | Joining Data Sources

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 63 joins can be in effect at one time, for a total of 256 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).

For details about data sources you can use in a join, see [Data Sources You Can and Cannot Join](#) on page 834.

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

Types of Joins

In this section:

Unique and Non-Unique Joined Structures

Recursive Joined Structures

Reference:

Data Sources You Can and Cannot Join

Notes on DBA Security for Joined Data Structures

When you join two data sources, some records in one of the files may lack corresponding records in the other file. When a report omits records that are not in both files, the join is called an inner join. When a report displays all matching records, plus all records from the host file that lack corresponding cross-referenced records, the join is called a left outer join.

The SET ALL command globally determines how all joins are implemented. If the SET ALL=ON command is issued, all joins are treated as outer joins. With SET ALL=OFF, the default, all joins are treated as inner joins.

Each JOIN command can specify explicitly which type of join to perform, locally overruling the global setting. This syntax is supported for FOCUS, XFOCUS, Relational, VSAM, IMS, and Adabas. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

You can also join data sources using one of two techniques for determining how to match records from the separate data sources. The first technique is known as an equijoin and the second is known as a conditional join. When deciding which of the two join techniques 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 845.

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 an employee ID in the host file to an 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 861.

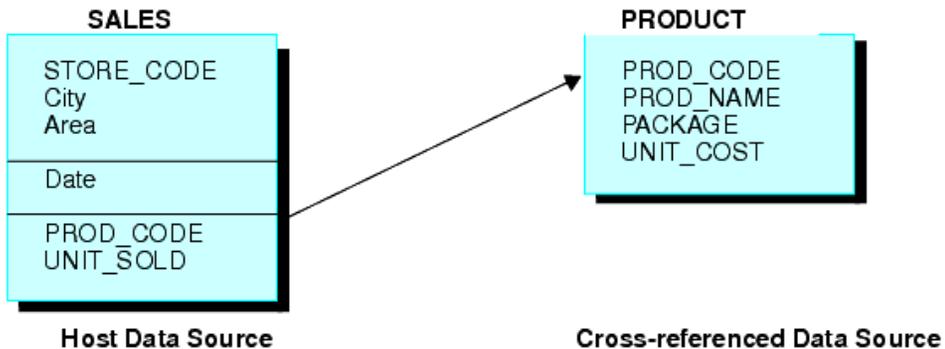
Joins can also be unique or non-unique. A unique, or one-to-one, join structure matches one value in the host data source to one value in the cross-referenced data source. Joining an employee ID in an employee data source to an employee ID in a salary data source is an example of a unique equijoin structure.

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 an employee ID in a company's employee data source to an employee ID in a data source that lists all the training classes offered by that company results 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 835.

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, thus saving space and making it easier to maintain product information. 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 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 835.
- ❑ 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 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**How to:**

Correct for Lagging Values With a Unique Join

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.

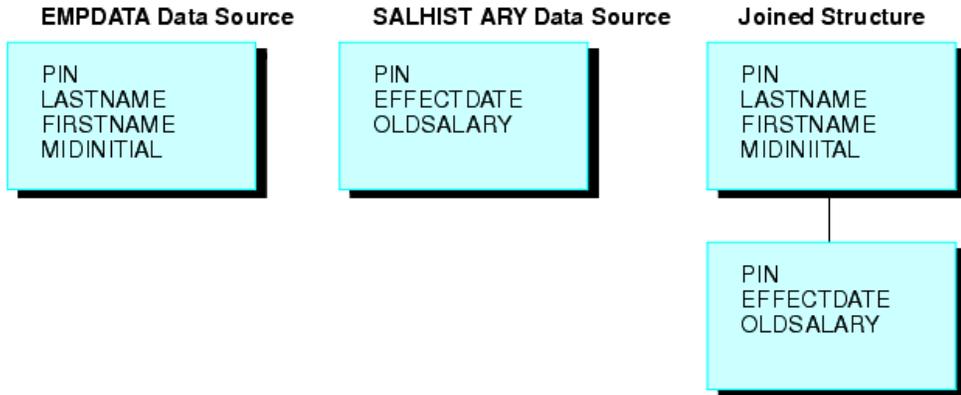
The ALL parameter in a JOIN command indicates that the joined structure is non-unique.

- ❑ Omit the ALL parameter only when you are sure that the joined structure is unique. Omitting the ALL parameter reduces overhead.
- ❑ The ALL parameter does 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 contains 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**

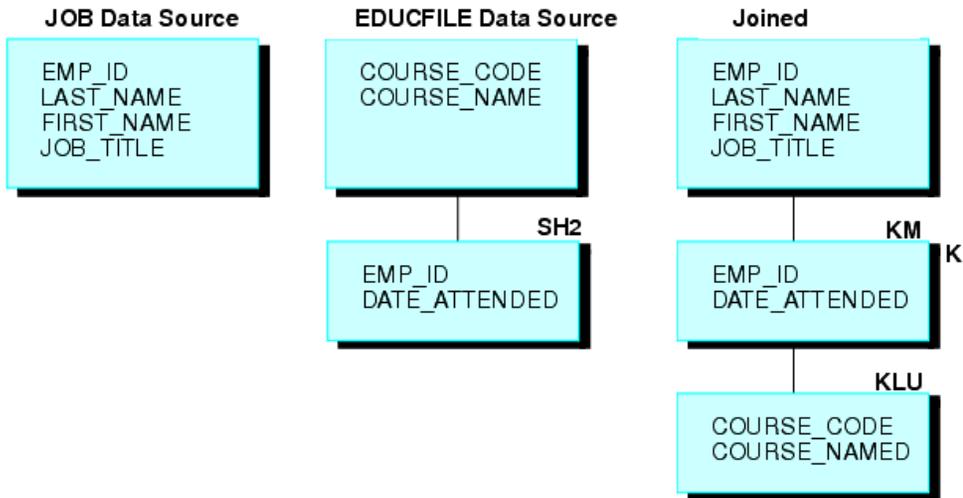
If a field value in the host file can appear in many segment instances in the cross-referenced file, then you should include the ALL phrase in the JOIN syntax. This structure is called a non-unique joined structure.

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 as many instances in the EDUCFILE data source 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
```



Syntax: How to Correct for Lagging Values With a Unique Join

If a parent segment has two or more unique child segments that each have multiple children, the report may incorrectly display a missing value. The remainder of the child values may then be misaligned in the report. These misaligned values are called lagging values. The JOINOPT parameter ensures proper alignment of your output by correcting for lagging values.

```
SET JOINOPT={NEW|OLD}
```

where:

NEW

Specifies that segments be retrieved from left to right and from top to bottom, which results in the display of all data for each record, properly aligned. Missing values only occur when they exist in the data.

OLD

Specifies that segments be retrieved as unique segments, which results in the display of missing data in a report where all records should have values. This might cause lagging values. OLD is the default value.

Example: Correcting for Lagging Values in a Procedure With Unique Segments and Multiple Children

This example is a hypothetical scenario in which you would use the JOINOPT parameter to correct for lagging values. Lagging values display missing data such that each value appears off by one line.

A single-segment host file (ROUTES) is joined to two files (ORIGIN and DEST), each having two segments. The files are joined to produce a report that shows each train number, along with the city that corresponds to each station.

The following request prints the city of origin (OR_CITY) and the destination city (DE_CITY). Note that missing data is generated, causing the data for stations and corresponding cities to lag, or be off by one line.

```
TABLE FILE ROUTES
PRINT TRAIN_NUM
OR_STATION OR_CITY
DE_STATION DE_CITY
END
```

The output is:

TRAIN_NUM	OR_STATION	OR_CITY	DE_STATION	DE_CITY
-----	-----	-----	-----	-----
101	NYC	NEW YORK	ATL	.
202	BOS	BOSTON	BLT	ATLANTA
303	DET	DETROIT	BOS	BALTIMORE
404	CHI	CHICAGO	DET	BOSTON
505	BOS	BOSTON	STL	DETROIT
505	BOS	.	STL	ST. LOUIS

Issuing SET JOINOPT=NEW enables segments to be retrieved in the expected order (from left to right and from top to bottom), without missing data.

```
SET JOINOPT=NEW
TABLE FILE ROUTES
PRINT TRAIN_NUM
OR_STATION OR_CITY
DE_STATION DE_CITY
END
```

The correct report has only 5 lines instead of 6, and the station and city data is properly aligned. The output is:

TRAIN_NUM	OR_STATION	OR_CITY	DE_STATION	DE_CITY
-----	-----	-----	-----	-----
101	NYC	NEW YORK	ATL	ATLANTA
202	BOS	BOSTON	BLT	BALTIMORE
303	DET	DETROIT	BOS	BOSTON
404	CHI	CHICAGO	DET	DETROIT
505	BOS	BOSTON	STL	ST. LOUIS

Recursive Joined Structures

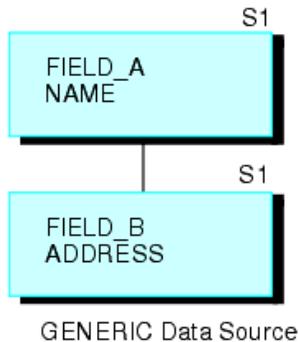
Reference:

Usage Notes for 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 is actually the case.

Example: Understanding Recursive Joined Structures

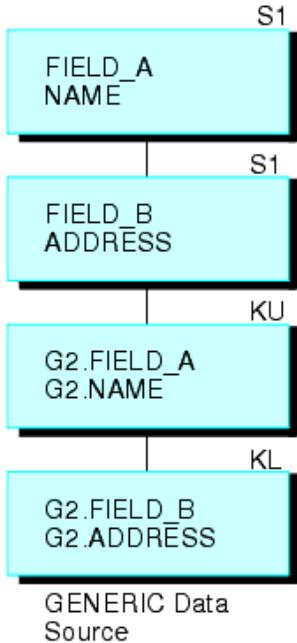
For example, the GENERIC data source shown below consists of Segments A and B.



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).



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 840.

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 856) 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.

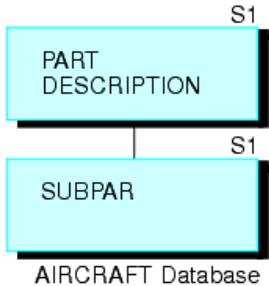
Example: Using Recursive Joined Structures

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 are repeated in every place they are used.

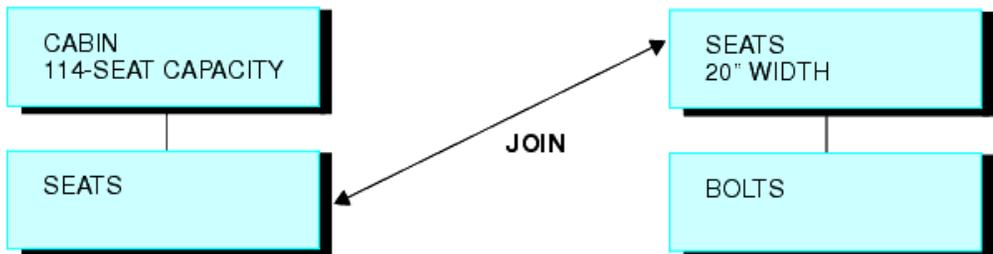
To reduce this repetition, design a data source that has only two segments (shown in the following diagram). The top segment describes each part of the aircraft, large and small. The bottom segment lists the component parts without descriptions.



Every part (except for 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 source.

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.

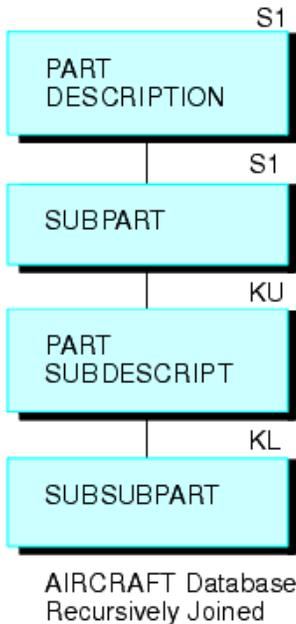


Sample Instance in the AIRCRAFT Data Source

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.



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 SUBPART BY PART BY SUBPART BY SUBDESCRIPT
END
```

How the JOIN Command Works

Reference:

Increasing Retrieval Speed in Joined Data Sources

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 63 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. For details about session boundaries in FOCUS, see the *Developing Applications* manual.

When two data sources are joined, one is called the host file; the other is called the cross-referenced file. Each time a record is retrieved from the host file, 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 fields in each data source with formats (character, numeric, or date) and values in common. The common formats ensure 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 (for example, 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 845.
- ❑ 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 856.
- ❑ A condition you specify in the JOIN command itself. See [How to Create a Conditional JOIN](#) on page 862.

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 [Improving Report Processing](#) on page 903.

Creating an Equijoin**In this section:**

Joining From a Virtual Field to a Real Field Using an Equijoin

Data Formats of Shared Fields

Joining Fields With Different Numeric Data Types

How to:

Join Real Fields

Reference:

Requirements for Cross-Referenced Fields in an Equijoin

Restrictions on Group Fields

Usage Notes for Inner and Outer JOIN Command Syntax

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 join may be a simple one based on one field in each file to be joined, or a multi-field join for data sources that support this type of behavior. The following syntax describes the simple and multi-field variations:

```
JOIN [LEFT_OUTER|INNER] hfld1 [AND hfld2 ...] IN hostfile [TAG tag1]
    TO [UNIQUE|MULTIPLE]
    crfield [AND crfld2 ...] IN crfile [TAG tag2] [AS joinname]
END
```

where:

```
JOIN hfld1
```

Is the name of a field in the host file containing values shared with a field in the cross-referenced file. This field is called the host field.

AND *hfld2...*

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 file is FOC. For example, assume the cross-referenced file contains a phone number field with 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.

INNER

Specifies an inner join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

LEFT OUTER

Specifies a left outer join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

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 the host file.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

TO [[UNIQUE](#)|[MULTIPLE](#)] *crfld1*

Is the name of a field in the cross-referenced file containing values that match those of *hfld1* (or of concatenated host fields). This field is called the cross-referenced field.

Note: Unique returns only one instance and, if there is no matching instance in the cross-referenced file, it supplies default values (blank for alphanumeric fields and zero for numeric fields).

Use the [MULTIPLE](#) parameter when *crfld1* may have multiple instances in common with one value in *hfld1*. Note that [ALL](#) is a synonym for [MULTIPLE](#), and omitting this parameter entirely is a synonym for [UNIQUE](#). See [Unique and Non-Unique Joined Structures](#) on page 835 for more information.

AND *crfld2...*

Is the name of a field in the cross-referenced file with values in common with *hfld2*.

Note: *crfld2* 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 840.

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 does not overwrite it.
- You want to clear it selectively later.
- The structure is recursive. See [Recursive Joined Structures](#) on page 839.

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 overwrites an existing join without a name.

END

Required when the JOIN command is longer than one line; it 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 JJOIN
```

Example: Creating an Inner Join

The following procedure creates three FOCUS data sources:

- ❑ EMPINFO, which contains the fields EMP_ID, LAST_NAME, FIRST_NAME, and CURR_JOBCODE from the EMPINFO segment of the EMPLOYEE data source.
- ❑ JOBINFO, which contains the JOBCODE and JOB_DESC fields from the JOBFILE data source.
- ❑ EDINFO, which contains the EMP_ID, COURSE_CODE, and COURSE_NAME fields from the EDUCFILE data source.

The procedure then adds an employee to EMPINFO named Fred Newman who has no matching record in the JOBINFO or EDINFO data sources.

```
TABLE FILE EMPLOYEE
SUM LAST_NAME FIRST_NAME CURR_JOBCODE
BY EMP_ID
ON TABLE HOLD AS EMPINFO FORMAT FOCUS INDEX EMP_ID CURR_JOBCODE
END
-RUN
```

```
TABLE FILE JOBFIL
SUM JOB_DESC
BY JOBCODE
ON TABLE HOLD AS JOBINFO FORMAT FOCUS INDEX JOBCODE
END
-RUN
```

```
TABLE FILE EDUCFILE
SUM COURSE_CODE COURSE_NAME
BY EMP_ID
ON TABLE HOLD AS EDINFO FORMAT FOCUS INDEX EMP_ID
END
-RUN
```

```
MODIFY FILE EMPINFO
FREEFORM EMP_ID LAST_NAME FIRST_NAME CURR_JOBCODE
MATCH EMP_ID
ON NOMATCH INCLUDE
ON MATCH REJECT
DATA
111111111, NEWMAN, FRED, C07,$
END
```

The following request prints the contents of EMPINFO. Note that Fred Newman has been added to the data source:

```
TABLE FILE EMPINFO
PRINT *
END
```

The output is:

EMP_ID	LAST_NAME	FIRST_NAME	CURR_JOBCODE
071382660	STEVENS	ALFRED	A07
112847612	SMITH	MARY	B14
117593129	JONES	DIANE	B03
119265415	SMITH	RICHARD	A01
119329144	BANNING	JOHN	A17
123764317	IRVING	JOAN	A15
126724188	ROMANS	ANTHONY	B04
219984371	MCCOY	JOHN	B02
326179357	BLACKWOOD	ROSEMARIE	B04
451123478	MCKNIGHT	ROGER	B02
543729165	GREENSPAN	MARY	A07
818692173	CROSS	BARBARA	A17
111111111	NEWMAN	FRED	C07

The following JOIN command creates an inner join between the EMPINFO data source and the JOBINFO data source.

```
JOIN CLEAR *
JOIN INNER CURR_JOBCODE IN EMPINFO TO MULTIPLE JOBCODE IN JOBINFO AS J0
```

Note that the JOIN command specifies a multiple join. In a unique join, the cross-referenced segment is never considered missing, and all records from the host file display on the report output. Default values (blank for alphanumeric fields and zero for numeric fields) display if no actual data exists.

The following request displays fields from the joined structure:

```
TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME JOB_DESC
END
```

Fred Newman is omitted from the report output because his job code does not have a match in the JOBINFO data source:

LAST_NAME	FIRST_NAME	JOB_DESC
STEVENS	ALFRED	SECRETARY
SMITH	MARY	FILE QUALITY
JONES	DIANE	PROGRAMMER ANALYST
SMITH	RICHARD	PRODUCTION CLERK
BANNING	JOHN	DEPARTMENT MANAGER
IRVING	JOAN	ASSIST.MANAGER
ROMANS	ANTHONY	SYSTEMS ANALYST
MCCOY	JOHN	PROGRAMMER
BLACKWOOD	ROSEMARIE	SYSTEMS ANALYST
MCKNIGHT	ROGER	PROGRAMMER
GREENSPAN	MARY	SECRETARY
CROSS	BARBARA	DEPARTMENT MANAGER

Example: Creating a Left Outer Join

The following JOIN command creates a left outer join between the EMPINFO data source and the EDINFO data source:

```
JOIN CLEAR *
JOIN LEFT_OUTER EMP_ID IN EMPINFO TO MULTIPLE EMP_ID IN EDINFO AS J1
```

The following request displays fields from the joined structure:

```
TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME COURSE_NAME
END
```

All employee records display on the report output. The records for those employees with no matching records in the EDINFO data source display the missing data character (.) in the COURSE_NAME column. If the join were unique, blanks would display instead of the missing data character.

LAST_NAME	FIRST_NAME	COURSE_NAME
STEVENS	ALFRED	FILE DESCRPT & MAINT
SMITH	MARY	BASIC REPORT PREP FOR PROG
JONES	DIANE	FOCUS INTERNALS
SMITH	RICHARD	BASIC RPT NON-DP MGRS
BANNING	JOHN	.
IRVING	JOAN	.
ROMANS	ANTHONY	.
MCCOY	JOHN	.
BLACKWOOD	ROSEMARIE	DECISION SUPPORT WORKSHOP
MCKNIGHT	ROGER	FILE DESCRPT & MAINT
GREENSPAN	MARY	.
CROSS	BARBARA	HOST LANGUAGE INTERFACE
NEWMAN	FRED	.

Example: Creating Two Inner Joins With a Multipath Structure

The following JOIN commands create an inner join between the EMPINFO and JOBINFO data sources and an inner join between the EMPINFO and EDINFO data sources:

```
JOIN CLEAR *
JOIN INNER CURR_JOBCODE IN EMPINFO TO MULTIPLE JOBCODE IN JOBINFO AS J0
JOIN INNER EMP_ID IN EMPINFO TO MULTIPLE EMP_ID IN EDINFO AS J1
```

The structure created by the two joins has two independent paths:

```

          SEG01
01      S1
*****
*EMP_ID      **I
*CURRE_JOBCODE**I
*LAST_NAME   **
*FIRST_NAME  **
*           **
*****
          I
          +-----+
          I           I
          I SEG01     I SEG01
02      I KM         03      I KM
.....
:EMP_ID      ::K  :JOBCODE      ::K
:COURSE_CODE ::  :JOB_DESC      ::
:COURSE_NAME ::  :              ::
:           ::  :              ::
:           ::  :              ::
:.....::  :.....::
.....:  :.....:
JOINED EDINFO  JOINED JOBINFO
    
```

The following request displays fields from the joined structure:

```

SET MULTIPATH=SIMPLE
TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME IN 12 COURSE_NAME JOB_DESC
END
    
```

With MULTIPATH=SIMPLE, the independent paths create independent joins. All employee records accepted by either join display on the report output. Only Fred Newman (who has no matching record in either of the cross-referenced files) is omitted:

LAST_NAME	FIRST_NAME	COURSE_NAME	JOB_DESC
-----	-----	-----	-----
STEVENS	ALFRED	FILE DESCRPT & MAINT	SECRETARY
SMITH	MARY	BASIC REPORT PREP FOR PROG	FILE QUALITY
JONES	DIANE	FOCUS INTERNALS	PROGRAMMER ANALYST
SMITH	RICHARD	BASIC RPT NON-DP MGRS	PRODUCTION CLERK
BANNING	JOHN	.	DEPARTMENT MANAGER
IRVING	JOAN	.	ASSIST.MANAGER
ROMANS	ANTHONY	.	SYSTEMS ANALYST
MCCOY	JOHN	.	PROGRAMMER
BLACKWOOD	ROSEMARIE	DECISION SUPPORT WORKSHOP	SYSTEMS ANALYST
MCKNIGHT	ROGER	FILE DESCRPT & MAINT	PROGRAMMER
GREENSPAN	MARY	.	SECRETARY
CROSS	BARBARA	HOST LANGUAGE INTERFACE	DEPARTMENT MANAGER

With MULTIPATH=COMPOUND, only employees with matching records in both of the cross-referenced files display on the report output:

LAST_NAME	FIRST_NAME	COURSE_NAME	JOB_DESC
STEVENS	ALFRED	FILE DESCRPT & MAINT	SECRETARY
SMITH	MARY	BASIC REPORT PREP FOR PROG	FILE QUALITY
JONES	DIANE	FOCUS INTERNALS	PROGRAMMER ANALYST
SMITH	RICHARD	BASIC RPT NON-DP MGRS	PRODUCTION CLERK
BLACKWOOD	ROSEMARIE	DECISION SUPPORT WORKSHOP	SYSTEMS ANALYST
MCKNIGHT	ROGER	FILE DESCRPT & MAINT	PROGRAMMER
CROSS	BARBARA	HOST LANGUAGE INTERFACE	DEPARTMENT MANAGER

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 XFOCUS 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 adds an index to your FOCUS or Fusion data source.

Note: The indexed fields can be external. See the *Describing Data* manual for more information about indexed fields and the Rebuild tool.

- ❑ 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 format or comma-delimited sequential files, any field can be a cross-referenced field. However, both the host and cross-referenced file must be retrieved in ascending order on the named (key) fields. If the data is not in the same sort order, errors are displayed. A many-to-many join is not supported. A comma-delimited file used as the cross-referenced file in the join must consist of only one segment. If the join is based on multiple fields, a fixed format sequential file must consist of a single segment. If the cross-referenced fixed format sequential 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.

Reference: Usage Notes for Inner and Outer JOIN Command Syntax

- ❑ The SET ALL and SET CARTESIAN commands are ignored by the syntax.
- ❑ The ALL. parameter is not supported. If the ALL. parameter is used, the following message displays:

`(FOC32452) Use of ALL. with LEFT_OUTER/INNER not allowed`
- ❑ If you define multiple joins, the resulting structure can be a single path or multi-path data source.

- ❑ If the setting MULTIPATH=SIMPLE is in effect and the report is based on multiple paths, each of the individual joins is constructed separately without regard to the other joins, and the matching records from each of the separate paths displays on the report output. Therefore, the output can contain records that would have been omitted if only one of the joins was in effect.
- ❑ If the setting MULTIPATH=COMPOUND is in effect with a multi-path report, or if the report displays data only from a single path, the report output displays only those records that satisfy all of the joins.

Joining From a Virtual Field to a Real Field Using an Equijoin

How to:

Join From a Virtual Field to a Real Field

Reference:

Notes on Using Virtual Fields With Joined Data Sources

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 865.

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 [LEFT_OUTER|INNER] deffld WITH host_field ...  
    IN hostfile [TAG tag1]  
    TO [UNIQUE|MULTIPLE]  
    cr_field IN crfile [TAG tag2] [AS joinname]  
END
```

where:

JOIN deffld

Is the name of a 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 857.

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.

The WITH phrase is required unless the KEEPDEFINES parameter is set to ON and *deffld* was defined prior to issuing the JOIN command.

To determine which segment contains the virtual field, use the ? DEFINE query after issuing the DEFINE command. See the *Developing Applications* manual for details about Query commands.

INNER

Specifies an inner join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

LEFT_OUTER

Specifies a left outer join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

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 JOIN commands of a joined structure.

TO [UNIQUE|MULTIPLE] *crfld1*

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.

Note: Unique returns only one instance and, if there is no matching instance in the cross-referenced file, it supplies default values (blank for alphanumeric fields and zero for numeric fields).

Use the MULTIPLE parameter when *crfld1* may have multiple instances in common with one value in *hffd1*. Note that ALL is a synonym for MULTIPLE, and omitting this parameter entirely is a synonym for UNIQUE. See [Unique and Non-Unique Joined Structures](#) on page 835 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 840.

The tag name for the host file must be the same in all 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 does 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 839.

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 overwrites 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 turn KEEPDEFINES ON to preserve the virtual fields. For more information, see [Preserving Virtual Fields During Join Parsing](#) on page 865.

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 in which 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 in which 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 and 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 DESCRPT & 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 DESCRPT & 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 alphanumeric, 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. For details, see [Joining Fields With Different Numeric Data Types](#) on page 860.

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 854.

If the host field is not a group field, the cross-referenced field can still 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

How to:

Enable Joins With Data Type Conversion

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**How to:**

Create 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.

Syntax: How to Create a Conditional JOIN

The syntax of the conditional (WHERE-based) JOIN command is

```
JOIN [LEFT_OUTER|INNER] FILE hostfile AT hfld1 [WITH hfld2] [TAG tag1]  
  
    TO {UNIQUE|MULTIPLE}  
    FILE crfile AT crfld [TAG tag2] [AS joinname]  
    [WHERE expression1;  
    [WHERE expression2;  
    ...]  
END
```

where:

INNER

Specifies an inner join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

LEFT_OUTER

Specifies a left outer join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

hostfile

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.

hfld1

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 is referenced.

tag1

Is the optional tag name that is used as a unique qualifier for fields and aliases in the host data source.

hfld2

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.

MULTIPLE

Specifies a one-to-many relationship between *from_file* and *to_file*. Note that ALL is a synonym for MULTIPLE.

UNIQUE

Specifies a one-to-one relationship between *from_file* and *to_file*. Note that ONE is a synonym for UNIQUE.

Note: Unique returns only one instance and, if there is no matching instance in the cross-referenced file, it supplies default values (blank for alphanumeric fields and zero for numeric fields).

crfile

Is the cross-referenced Master File.

crfld

Is the join field name in the cross-referenced Master File. It can be any field in the segment.

tag2

Is the optional tag name that is used as a unique qualifier for fields and aliases in the cross-referenced data source.

joinname

Is the name associated with the joined structure.

expression1, expression2

Are any expressions that are acceptable in a DEFINE FILE command. All fields used in the expressions 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 Transaction	Release Date	Years 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

In this section:

Preserving Virtual Fields Using KEEPDEFINES

Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN

Screening Segments With Conditional JOIN Expressions

Parsing WHERE Criteria in a Join

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

How to:

Use KEEPDEFINES

Reference:

Usage Notes for 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 removes 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.

Reference: **Usage Notes for KEEPDEFINES**

Virtual fields defined prior to setting KEEPDEFINES ON are not preserved after a JOIN command.

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 a 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
COMPUTE ACTUAL_DAYS/I2 = RETURNDATE-TRANSDATE;
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 number of actual 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 [Creating Temporary Fields](#) on page 205.

Example: Preserving Virtual Fields With 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
END
? 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 on which they reside (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 used simply as segment references.

Note: If no WHERE criteria are in effect, you receive a Cartesian product.

Displaying Joined Structures

How to:

Display a Joined Structure

List 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 871.

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 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

In this section:

Clearing a Conditional Join

How to:

Clear a Join

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 using the same host file are also 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 873.

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 is cleared. Any virtual fields saved in the context of a join that is cleared are also cleared. Normal joins may or may not be cleared, depending on the position of the conditional join. The JOIN CLEAR * command clears every join issued, along with its associated virtual fields. However, all virtual fields in the null context 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 JW2
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 JW3
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 JW4
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 Joined Structures

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

17

Merging Data Sources

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.

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

Merging Data

How to:

Merge Data Sources

Reference:

Usage Notes for Match Requests

You can merge two or more data sources, and specify which records to merge and which to sort 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 use any other HOLD file. The merge process does not change the original data sources. For more information on HOLD files, see [Saving and Reusing Your Report Output](#) on page 421

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 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 16 sets of data in one Match request. For example, you can merge different data sources, or data from the same data source.

Note: The limit of 16 applies to the most complex request. Simpler requests may be able to merge more data sources.

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 882.

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 877.

Reference: Usage Notes for Match Requests

- ❑ 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 *Displaying Report Data* on page 45.

- ❑ 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 parameter behaves as if HOLDLIST were set to ALL.
- ❑ You cannot use BY HIGHEST in a MATCH request.
- ❑ The following prefix operators are not supported in MATCH requests: DST., DST.CNT., RNK., ST., and CT.

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 appear in the HOLD file.

Note that if you 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:

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

How to:

Specify Merge Phrases

Reference:

Merge Phrases

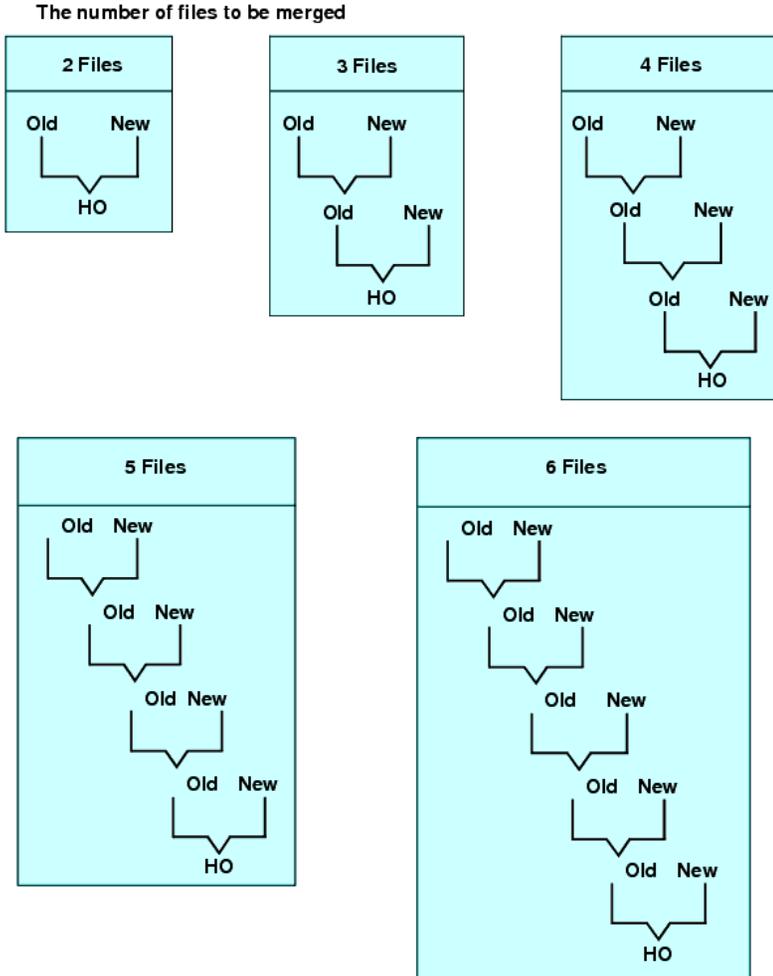
The way MATCH merges data depends on the order in which you name data sources in the request, the BY fields, display commands, and the merge phrases you use. In general, however, processing is as follows:

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 882.
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:



Syntax: **How to Specify Merge Phrases**

```
AFTER MATCH HOLD [AS 'name'] mergetype
```

where:

AS 'name'

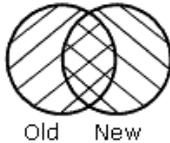
Specifies the name of the extract data source created by the MATCH command. The default is HOLD.

mergetype

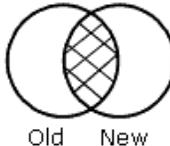
Specifies how the retrieved records from the files are to be compared.

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.

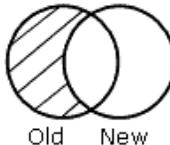
OLD-OR-NEW specifies that all records from both the old data source and the new data source appear in the HOLD file. This is the default if the AFTER MATCH line is omitted.



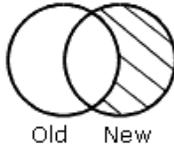
OLD-AND-NEW specifies that records that appear in both the old and new data sources appear in the HOLD file. (The intersection of the sets.)



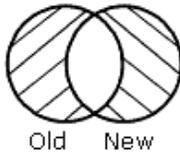
OLD-NOT-NEW specifies that records that appear only in the old data source appear in the HOLD file.



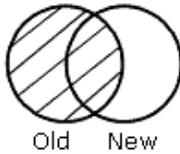
NEW-NOT-OLD specifies that records that appear only in the new data source appear in the HOLD file.



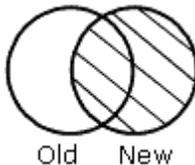
OLD-NOR-NEW specifies that only records that are in the old data source but not in the new data source, or in the new data source but not in the old, appear in the HOLD file (the complete set of non-matching records from both data sources).



OLD specifies that all records from the old data source, and any matching records from the new data source, are merged into the HOLD file.



NEW specifies that all records from the new data source, and any matching records from the old data source, are 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 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	212289111	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 records from the EMPLOYEE and EDUCFILE files.
- ❑ There are records from both files with an EMP_ID value of 071382660. Since there is a match, this record is written to the HOLD file:

```
Record n: 071382660 STEVENS 101
```

- ❑ There are records from both files with an EMP_ID value of 112847612. Since there is a match, this record is written to the HOLD file:

Record n: 112847612 SMITH 103

- ❑ The records do not match where a record from the EMPLOYEE file has an EMP_ID value of 119329144 and a record from the EDUCFILE file has an EMP_ID value of 212289111. The record with the lower value is written to the HOLD file and a space is inserted for the missing value:

Record n: 119329144 BANNING

- ❑ Similarly, the 212289111 record exists only in the EDUCFILE file, and is written as:

Record n: 212289111 103

The following code produces a report of the records in the HOLD file:

```
TABLE FILE HOLD
PRINT *
END
```

The output is:

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:

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 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 is included.

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

In this section:

Field Name and Format Matching

How to:

Concatenate Data Sources

With universal concatenation, you can retrieve data from unlike data sources 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, and 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 [Selecting Records for Your Report](#) on page 157.

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.

```
{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. *Master Files and Diagrams* on page 1113, contains the Master Files referenced in the request.

Tip: PAYHIST is a fixed-format file. You need to issue a FILEDEF or ALLOCATE command in order to use it. See the *Overview and Operations* manual for more information.

```

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

```

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:

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

In this section:

Using Sort Fields in MATCH Requests

How to:

Merge Concatenated Data Sources

You can use the MORE phrase in a MATCH request to merge up to 16 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 in 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**

```
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 first page of output is:

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 first page of output is:

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

How to:

Enable/Disable Cartesian Product

Reference:

Usage Notes for 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 generates 16 records. When CARTESIAN is OFF, the same request generates only three records.

For related information about controlling how selection tests are applied to child segments on independent paths, see [Selecting Records for Your Report](#) on page 157.

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 parameter 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 parameter, and no warning message is issued.
- ❑ MATCH is not supported with the SET CARTESIAN parameter. 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 parameter.

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 in an EBCDIC environment is:

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 in an EBCDIC environment is:

CAR	MODEL	STANDARD
JAGUAR	V12XKE AUTO	POWER STEERING
	XJ12L AUTO	RECLINING BUCKET SEATS
	.	WHITEWALL RADIAL PLY TIRES
	.	WRAP AROUND BUMPERS
	.	4 WHEEL DISC BRAKES

18 | Improving Report Processing

The following high-performance methods optimize data retrieval and report processing:

- ❑ Temporary rotation of network and hierarchical data sources to create an alternate view of the data.
- ❑ Automatic alternate file views with the AUTOPATH feature.
- ❑ Automatic indexed retrieval (AUTOINDEX).
- ❑ Retrieval of pre-sorted data using the TABLEF command.
- ❑ Preserving the internal matrix of a report using the SAVEMATRIX parameter.
- ❑ Compiling expressions into machine code to provide faster processing.
- ❑ The Pooled Tables option to produce many reports or extract files in a single pass of your data source, reducing database I/O, CPU, and elapsed time.

Note: These techniques may not be available for all data sources. See your data adapter documentation to determine if a technique is valid for your data source.

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
- ❑ Compiling Expressions
- ❑ Producing Multiple Outputs in One Pass of a Data Source (Pooled Tables)

Rotating a Data Structure for Enhanced Retrieval

How to:

Request an Alternate View

Reference:

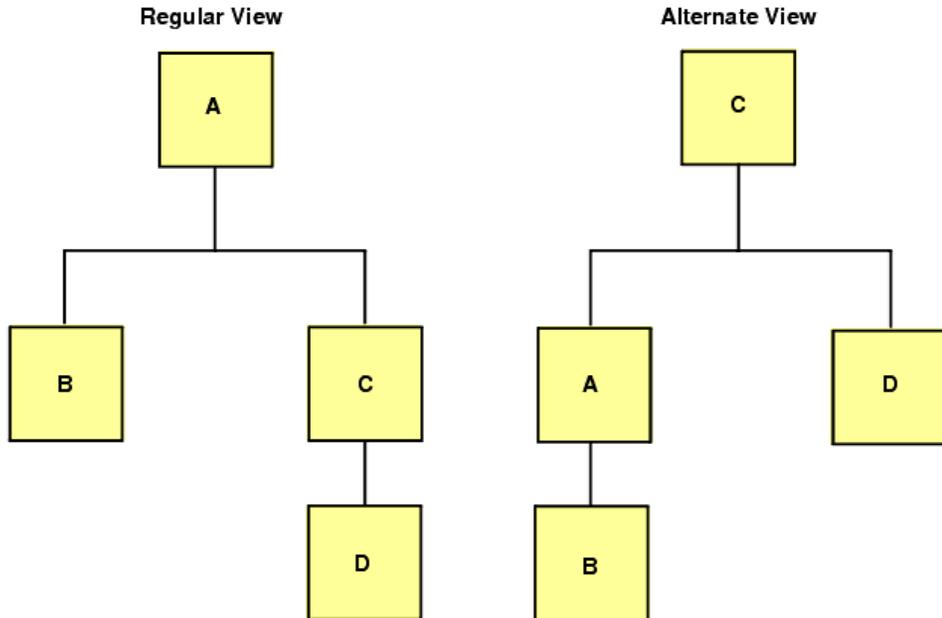
Usage Notes for Restructuring Data

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.

It should be noted that retrieval is controlled by the minimum referenced subtree. For more information, see *Understanding the Efficiency of the Minimum Referenced Subtree* in the *Describing a Group of Fields* chapter in the *Describing Data* manual.

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 found in the alternate root segment to the file name in the TABLE command, separated by a period (.):

```
TABLE FILE filename.fieldname
```

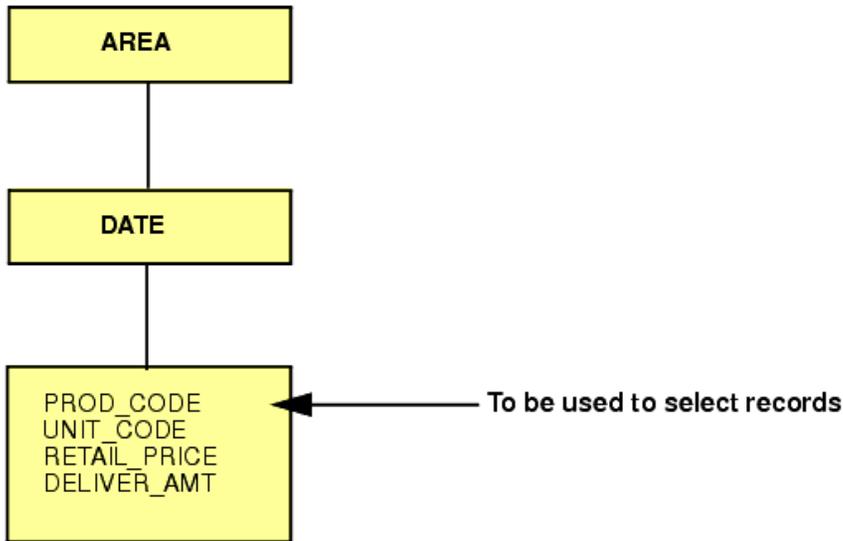
Reference: **Usage Notes for Restructuring Data**

- ❑ If you use a non-indexed field, each segment instance is retrieved until the specified record is found. Therefore, this process is 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 is retrieved in physical order, not logical order. This is 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

How to:

Use Indexed Retrieval

Reference:

Usage Notes for Indexed Retrieval

Automatic indexed retrieval (AUTOINDEX) optimizes the speed of data retrieval in FOCUS 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.

Reference: **Usage Notes for Indexed Retrieval**

- ❑ AUTOINDEX is never invoked when the TABLE request contains an alternate file view (that is, TABLE FILE *filename.fieldname*).
- ❑ Even if AUTOINDEX is ON, indexed retrieval is not performed when the TABLE request contains BY HIGHEST or BY LOWEST phrases.
- ❑ When a request specifies an indexed view (as in TABLE FILE *filename.indexed_fieldname*), indexed retrieval is implemented under the following circumstances:
 - ❑ AUTOINDEX is OFF and the request contains an equality test on the indexed field.
 - ❑ AUTOINDEX is ON and the request contains either an equality or a range (FROM ... TO) test against the indexed field.

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 requires no additional sorting.

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 it were set to ON.
- ❑ The DST. prefix operator is not permitted.
- ❑ BORDER styling is not supported with TABLEF.

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**How to:**

Save an Internal Matrix

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 overwritten 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. OFF is the default value.

Example: Saving the Internal Matrix of a Report

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
```

Compiling Expressions

In this section:

- Compiling Expressions Using the DEFINES Parameter
- Compiling Expressions Using the COMPUTE Parameter

Compiling expressions into machine code provides faster processing.

Compiling Expressions Using the DEFINES Parameter

How to:

- Compile DEFINE Expressions
- Query Compiled DEFINE Expressions

On z/OS and z/VM, two expression compilers are available. By issuing the appropriate command, you can select one of them or disable compilation of expressions. Both compilers cannot be active for the same request:

- ❑ The DEFINE compiler compiles only those expressions that are found in DEFINE fields referenced in TABLE requests, but it provides much faster execution of those expressions than the other compiler. It compiles expressions using the arithmetic operations built into the underlying operating system, and is, therefore, referred to as the native compiler. Compilation takes place at TABLE run time. This compiler is invoked by issuing the command SET DEFINES = COMPILED.

- ❑ The other compiler is invoked with the command SET COMPUTE = NEW. This compiler provides expression compilation for DEFINE, IF, and WHERE commands in TABLE procedures. Under this compiler, expressions are compiled at DEFINE time. Therefore, compilation may be invoked for expressions that are never actually used in a request.

Among the benefits of the DEFINE compiler are:

- ❑ Compilation of only those expressions that are actually used in the TABLE request.
- ❑ Much faster execution of expressions containing complex calculations on long packed fields.
- ❑ Compilation of date expressions.

After the native DEFINE compiler is invoked, any request that uses a DEFINE expression causes the expression to be compiled and then loaded into the system. For each record of the request that needs computation, the system executes the generated code. This compiler is most effective with TABLE requests that include a large number of DEFINE fields and read a large number of records because the speed of evaluation per record in such requests offsets the extra compilation and load steps.

Note: To compile expressions in MODIFY procedures in Mainframe environments, use the SET MODCOMPUTE command.

Syntax: How to Compile DEFINE Expressions

Issue the following command FOCPARM, a FOCEXEC, or at the command line:

```
SET DEFINES = {COMPILED|OLD}
```

where:

COMPILED

Implements expression compilation at request run time, compiling only those DEFINES that are used in the request. COMPILED is the default value.

OLD

Leaves expression compilation up to the control of the current SET COMPUTE value. If you issue the SET DEFINES = OLD command, the COMPUTE parameter is automatically set to NEW.

Syntax: How to Query Compiled DEFINE Expressions

Issue the following command to query the current setting:

```
? SET DEFINES
```

Compiling Expressions Using the COMPUTE Parameter

How to:

Control Expression Compilation Using the COMPUTE Parameter

Reference:

Usage Notes for SET COMPUTE

Interaction Between SET DEFINES and SET COMPUTE

Usage Notes for Compiled DEFINES

The compiler implemented with the SET COMPUTE = NEW command provides expression compilation for DEFINE, IF, and WHERE commands in TABLE procedures. Under this compiler, expressions are compiled at DEFINE time. Therefore, compilation may be invoked for expressions that are never actually used in a request.

Syntax: **How to Control Expression Compilation Using the COMPUTE Parameter**

```
SET COMPUTE = {NEW|OLD|NATV}
```

where:

NEW

Compiles DEFINE calculations when a request is executed.

OLD

Does not compile DEFINE calculations when a request is executed. The old logic is used.

[NATV](#)

Compiles DEFINE calculations using the native compiler. This setting is also activated by the SET DEFINES=COMPILED command, which is the default setting.

Reference: **Usage Notes for SET COMPUTE**

The following calculations are not compiled with SET COMPUTE = NEW:

- ❑ 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).

Reference: Interaction Between SET DEFINES and SET COMPUTE

Two expression compilers are available, but only one can be activated for any request. Activating either compiler automatically deactivates the other compiler:

- ❑ Issuing the SET DEFINES=COMPILED command activates the new compiler and deactivates the old compiler by automatically setting the value of the COMPUTE parameter to NATV (native compiler).
- ❑ Issuing the SET DEFINES=OLD command deactivates the new compiler and automatically activates the old compiler by automatically setting the value of the COMPUTE parameter to NEW .
- ❑ Issuing the SET COMPUTE command with either the OLD or NEW setting deactivates the new compiler. The OLD setting also deactivates the old compiler.

Therefore, you can select either compiler by issuing the SET DEFINES command. DEFINES=COMPILED selects the new compiler, DEFINES=OLD selects the old compiler. To turn compilation off, issue SET COMPUTE=OLD.

The new compiler is recommended for TABLE requests that include a large number of DEFINE fields (especially those that use packed arithmetic or date expressions) and read a large number of records.

If a TABLE request retrieves a large number of records or if the DEFINE fields use packed arithmetic (especially with long packed fields) or date expressions, the new compiler is likely to provide the most benefit.

Reference: Usage Notes for Compiled DEFINES

- ❑ Any expression that cannot be compiled runs without compilation. This does not affect compilation of other expressions. The following elements in an expression disable compilation with the new compiler:
 - ❑ Functions. However, expressions that use the following functions can be compiled: YMD, DMY, INT, and DECODE.
 - ❑ CONTAINS, OMMITS, LAST.
- ❑ The SET DEFINES command is not supported in an ON TABLE phrase.
- ❑ SET DEFINES creates a pool boundary when used in conjunction with Pooled Tables.

If compilation is not possible because of environmental conditions, the processing is handled without compilation. No message is generated indicating that compilation did not take place. To determine whether it did take place, issue the ? COMPILE command.

Producing Multiple Outputs in One Pass of a Data Source (Pooled Tables)

In this section:

Overview

Sub-Pool Boundaries and Pooling Restrictions

Estimating Memory Requirements

Memory Requirements

Sharing Selection Criteria and Filters Across Requests in a Pool

Criteria When Pooling Non-Relational Database Requests

Criteria When Pooling Relational Database Requests

Criteria When Pooling Batch Requests

Selecting a Sort Utility

Observing the Results of Pooling (TRACEON)

Installing the Pooled Tables Option

The Pooled Tables option permits you to produce many reports or extract files in a single pass of your data source, dramatically reducing database I/O, CPU, and elapsed time. Requests against any data source, file, or JOIN structure that FOCUS reads can be pooled without incurring a penalty, even if the application does not exploit the feature.

Pooling is added with several SET commands, and its analytical functions can automatically identify reports that can share database I/O and run them concurrently.

Pooling is applicable whenever consecutive report requests run against the same database, which is ideal for large batch operations, as well as canned FOCUS reporting and data-extract applications. It also applies in most reporting situations where record-selection costs exceed the costs for report formatting.

Overview

How to:

Activate the Pooled Tables Feature

To implement Pooled Tables in an application, you simply add several SET commands; no other changes are required. As FOCUS runs a group of report requests, it starts pooling as soon as it encounters a SET POOL=ON command, and pooling continues until it reads a SET POOL=OFF. During processing, FOCUS searches for consecutive TABLE requests that access the same data source with the same access method, and it stores those in sub-pools. A read-ahead feature even crosses FOCEXEC boundaries, dividing commands into retrieval and non-retrieval sub-pools. These sub-pools are collections of TABLE requests and related commands against common data sources—only report requests within sub-pools can be combined. Sub-pool boundaries are established whenever FOCUS encounters commands that either alter the data or change the processing environment (see [Sub-Pool Boundaries and Pooling Restrictions](#) on page 919).

Sub-pools are further subdivided into clusters, which are sets of consecutive TABLE requests against the same logical database that employ the same access method. Requests that cannot be pooled due to syntactical or environmental conditions are executed as single-TABLE clusters, which execute concurrently and share their data retrieval and screening processes (as well as related overhead), but do not share sorts or output formatting functions.

You can make processing more efficient if you can estimate the expected number of records to be read and lines of output. You can also perform a degree of memory management by limiting the amount of memory made available for pooling. For more information, see [Estimating Memory Requirements](#) on page 922

Syntax: **How to Activate the Pooled Tables Feature**

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

where:

OFF

Ends Pooled Tables and executes any queued requests. OFF is the default value.

ON

Activates Pooled Tables.

Example: Using Pooled Tables

The following example illustrates the ease of implementing Pooled Tables. Here a small amount of memory is provided for Pooled Tables (4,000K); then pooling is turned on and report size estimates are provided for each report. The report requests are queued until pooling is turned off. At that time, data is retrieved only once for all report requests in the pool. They are executed concurrently, and the reports printed one after the other.

```
SET POOLMEMORY = 4000
SET POOL=ON
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE BY DEPARTMENT
IF HIRE_DATE GE 820101
ON TABLE SET ESTLINES 10000 AND ESTRECORDS 10000
END
```

The output is:

DEPARTMENT	LAST_NAME	FIRST_NAME	HIRE_DATE
MIS	JONES	DIANE	82/05/01
	BLACKWOOD	ROSEMARIE	82/04/01
	GREENSPAN	MARY	82/04/01
PRODUCTION	SMITH	RICHARD	82/01/04
	BANNING	JOHN	82/08/01
	IRVING	JOAN	82/01/04
	ROMANS	ANTHONY	82/07/01
	MCKNIGHT	ROGER	82/02/02

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY CURR_JOBCODE
IF CURR_JOBCODE EQ 'A$*'
ON TABLE SET ESTLINES 5 AND ESTRECORDS 4000
END
```

The output is:

CURR_JOBCODE	CURR_SAL
A01	\$9,500.00
A07	\$11,000.00
	\$9,000.00
A15	\$26,862.00
A17	\$29,700.00
	\$27,062.00

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME BY DEPARTMENT
IF PAY_DATE FROM 820701 TO 820831
ON TABLE SET ESTRECORDS 120000
END

SET POOL=OFF
```

The output is:

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

Sub-Pool Boundaries and Pooling Restrictions

Reference:

Retrieval Commands Included in Sub-pools
 Commands That Cause Sub-Pool Boundaries
 SET Commands That Cause Sub-Pool Boundaries
 Restrictions for Single TABLE Clusters

Sub-pools are collections of TABLE or GRAPH requests and related commands. Only reports within a sub-pool can be pooled together to share I/O. Sub-pool boundaries are imposed by non-retrieval commands that can change the data or retrieval method for the data source. Therefore, you cannot reliably pool together reports on either side of a sub-pool boundary. When sub-pool boundary commands are encountered, pooling temporarily halts and all queued requests are executed.

Sub-pool boundaries are created when:

- ❑ A FOCEXEC completes execution and control is returned to the command line.
- ❑ A -RUN or -EXIT command is issued in a FOCEXEC.
- ❑ A DEFINE FILE *filename* ADD command is issued.
- ❑ A non-TABLE or GRAPH command is issued that could change the data (MAINTAIN, MODIFY, SQL), change the source of the data (DYNAM, USE), change the retrieval method (JOIN, PASS, FILTER), or change the operating environment (TSO, z/OS, CMS).
- ❑ Any SET or ON TABLE SET command that can alter retrieval or the Pooled Tables environment.

Reference: Retrieval Commands Included in Sub-pools

The following table lists retrieval commands included in sub-pools.

Note: This list may be subject to change in future releases.

?	?F	?FF	CHECK	DEFINE
GRAPH	HELP	HOLD	OFFLINE	ONLINE
PCHOLD	RELOT	RETYPE	SAVB	SAVE
TABLE	TABLEF			

Reference: Commands That Cause Sub-Pool Boundaries

The following table lists commands that cause sub-pool boundaries.

ANALYSE	CALC	CMS	COMBINE	
COMPILE	CREATE	DECRYPT	DYNAM	ENCRYPT
EX	EXEC	FILETALK	FILTER	FIN
FINISH	FIXPACK	FS	FSCAN	GRAPHTALK
JOIN	LET	LOAD	MAINTAIN	MATCH
MODIFY	MODIFYTALK	MPAINT	MVS	PASS
PLOTTALK	REBUILD	RECALC	REMOTE	RESTRICT
RUN	SCAN	SET	SQL	TABLETALK
TED	TSO	UNLOAD	USE	WINDOW

Reference: SET Commands That Cause Sub-Pool Boundaries

The following lists SET commands that cause sub-pool boundaries. SET commands included in ? SET ALL that are not on this list do not cause sub-pool boundaries.

ADABAS	AGRRATIO	ALL.	AUTOINDEX
AUTOPATH	AUTOSTRATEGY	AUTOTABLEF	BINS
BLKCALC	BYPANEL 2	BYSCROLL	CACHE
CALC	CALCMEMORY	CALCROWS	CALCWAIT
CARTESIAN	CDN	COLUMNSCROLL2	COMMIT
COMPUTE	CURRENCY	DATETIME	
DEFCENT	ESTLINES 1	ESTRECORDS 1	EXTSORT
FIELDNAME	FILENAME	FIXRETRIEVE	FOCSTACK
FOC144 1	HIPERFOCUS	HTMLMODE	ICUFORM
IMPLIEDLOAD	IMS	LABELPROMPT	LANGUAGE
LE370	LOADLIMIT	LOOKGRAPH	MAXLRECL
MAXPOOLMEM	MINIO	MODE XXXXXX	MPRINT
PASS	POOL	POOLBATCH	POOLFEATURE
POOLMEMORY	POOLRESERVE	PREFIX	PREVIEW
PRINTPLUS 2	QUALCHAR	RECORDLIMIT 1	SAVEMATRIX
SHIFT	SM	SQLENGINE	SQLTCARTES
SQLTOPTTF	STYLEMODE	SUSI	SUTABSIZE
TCPIPINT	TEMP DISK	TERMINAL	TEXTFIELD
TRACKIO	TRMSD	TRMSW	TRMTYP
USER	WINPFKEY	XRETRIEVAL	YRTHRESH
3DGRAPH			

Note:

- ❑ 1 indicates a sub-pool boundary with SET only.
- ❑ 2 indicates a sub-pool boundary with ON TABLE SET only.

Reference: Restrictions for Single TABLE Clusters

In certain instances, reports cannot be pooled due to syntactical or environmental conditions. In this case, they are executed as single TABLE clusters. Reports in this category include:

- ❑ TABLEF requests.
- ❑ MATCH requests.
- ❑ Extended Matrix Reports (EMRs).
- ❑ Reports using SET ALL=ON or PASS.

- ❑ Reports against FOCUS databases using an explicit indexed view or an implicit indexed view, using AUTOINDEX.
- ❑ Reports against relational databases where aggregation is passed to the DBMS.
- ❑ Reports that use MORE, ON field RECAP, COUNT DISTINCT, DST., INCLUDES, EXCLUDES, or COUNT as a verb object.
- ❑ Reports that use a redefined database field.
- ❑ Reports issued from the FOCUS command line.
- ❑ Reports that use a self-referential Filter or DBA value restriction.
- ❑ Reports that have more than 256 values in an equality IF or WHERE test.
- ❑ Reports executed when \$ORTPARM is allocated.
- ❑ Reports that use a user-written subroutine except those in Table 4. Generally, subroutines that require initialization and are then reused cannot be pooled. Random-number generator subroutines are a good example.

The list below provides subroutines and functions that can be pooled.

ARGLEN	ATODBL	AYM	AYMD	BAR
BITSON	BITVAL	BYTVAL	CHGDAT	CHKFMT
CHKPCK	CTAN	CTRFLD	DADMY	DADYM
DAMDY	DAMYD	DAYDM	DAYMD	DMOD
DOWK	DOWKL	DTDYD	DTDYM	DTMDY
DTMYD	DTYDM	DTYMD	EXP	FEXERR
FINDMEM	FMOD	FTOA	GETPDS	GETTOK
GETUSER	GREGDT	HEXBYT	HHMSS	IMOD
ITONUM	ITOPACK	ITOZ	JULDAT	LCWORD
LJUST	LOCASE	OVLAY	PARAG	PCKOUT
POSIT	RJUST	SOUNDEX	SUBSTR	TODAY
UFMT	UPCASE	YM		

Estimating Memory Requirements

The number of executable reports per cluster depends on how much memory is allocated to Pooled Tables (POOLMEMORY). To optimize pooling capacity, give POOLMEMORY an adequate size: z/OS region size, or z/VM virtual memory. Reduce POOLRESERVE after loading interface and other modules.

To improve the pooling potential of requests, remove unnecessary sub-pool boundary commands such as extraneous -RUN statements, and consolidate necessary sub-pool boundary-forcing commands such as DYNAM and SET. You can further improve opportunities for pooling within clusters by grouping requests with the same source, entry point and retrieval method.

For optimal processing, supply accurate estimates for ESTRECORDS, ESTLINES, and POOLMEMORY for each request. Remember that these estimates apply to each report, not to the aggregate size of the set of reports in the cluster. To calculate these sizes, issue ? STAT and review the statistics.

Example: Displaying Report Statistics

The statistical report produced by ? STAT is useful in tuning applications that employ Pooled Tables.

Note: This annotated sample shows only information concerning Pooled Tables.

```

                                STATISTICS OF LAST COMMAND
RECORDS      =          50000      .
LINES        -          50000      .
.
.
.
1.  READS      =        250000      .
.
.
.
2.  SUBPOOL    =           1        .
3.  CLUSTER    =           2  8.  ITERATION      =           1
4.  # CLUSTER ITEMS =        25  9.  # ITER ITEMS      =        16
5.  SEQ# IN CLUSTER =           5  10. SEQ# IN ITER      =           5
6.  ESTIMATED RECS =    50000  11. ESTIMATED LINES      =    50000
7.  REPORT WIDTH =        148

```

For this report:

1. READS Total number of read I/O's.
2. SUB POOL This is the first subpool.
3. CLUSTER This is the second cluster in the subpool.
4. # CLUSTER ITEMS Total number of reports in the cluster. There are 25 reports in the cluster.
5. SEQ # IN CLUSTER Sequence number of the current report in the cluster. This is the fifth report in the cluster.
6. ESTIMATED RECS ESTRECORDS was set to 50,000. Compare this with RECORDS at the top of the output. If a discrepancy, exists correct ESTRECORDS.
7. REPORT WIDTH The report width is 148 bytes.
8. ITERATION Sequence number of the iteration (multiple iterations are used if pool memory is insufficient to run all reports at once). This report was produced in the first iteration.
9. # ITER ITEMS Number of reports within the iteration. Sixteen reports are included in the first iteration.
10. SEQ # IN ITER Sequence number of this report within the iteration. This is the fifth report in iteration number one.
11. ESTIMATED LINES ESTLINES was set to 50,000. Compare this with LINES at the top of the output. If a discrepancy exists, correct your ESTLINES value for this report.

Memory Requirements

How to:

Supply an Estimate for the Number of Input Records for a Report

Supply an Estimate of the Number of Output Lines Expected for a Report

Limit the Amount of Memory Available for Pooling Within a Cluster (Per User)

Reserve Memory for Other Modules

Pooled Tables memory requirements per report vary depending on numbers of records selected, output lines produced, and report widths, all of which Pooled Tables calculates based on the values of ESTLINES and ESTRECORDS. Gather ESTLINES and ESTRECORDS input from:

- The statistical message: NUMBER OF RECORDS IN TABLE= LINES=.
- The RECORDS and LINES information available on the ? STAT output.

- ❑ Previously gathered information from the &RECORDS and &LINES variables.
- ❑ When using ACROSS, ESTLINES is the number of lines times the number of unique ACROSS columns.
- ❑ When using IF TOTAL or WHERE TOTAL, ESTLINES is the number of lines before the TOTAL selection is made.

The memory requirement for a small summary report roughly equals:

`NUMBER OF LINES OF OUTPUT * REPORT WIDTH.`

For large summary reports and detail reports, use:

`NUMBER OF RECORDS SELECTED * REPORT WIDTH`

In the absence of estimates, Pooled Tables uses the following defaults:

- ❑ ESTRECORDS=100000.
- ❑ ESTLINES=0.
- ❑ POOLMEMORY=16,384K.
- ❑ POOLRESERVE=100K(z/OS)/1024K (z/VM).

Syntax: How to Supply an Estimate for the Number of Input Records for a Report

`ON TABLE SET ESTRECORDS {m|0}`

where:

m

Is the estimate of the number of records being retrieved for a report. The default value is 0.

Assign a global value for each report in a pool with the following command:

`SET ESTRECORDS={m|0}`

Syntax: How to Supply an Estimate of the Number of Output Lines Expected for a Report

`ON TABLE SET ESTLINES nSET ESTLINES=n`

where:

n

Is a user estimate of the number of output lines for a report. The default value is 0. If no value is given, Pooled Tables assumes there is no aggregation, and that the number of lines is the same as the number of records.

You can assign a global value for each report in a pool with the following command:

```
SET ESTLINES=n
```

When ESTLINES is 0, Pooled Tables uses the current value of ESTRECORDS for ESTLINES. While adequate for large extract reports, this provides minimal benefit if inaccurate.

Syntax: **How to Limit the Amount of Memory Available for Pooling Within a Cluster (Per User)**

```
SET POOLMEMORY = n
```

where:

n

Is the upper limit in kilobytes of memory that FOCUS may use during any cluster for a user. In z/OS, this is memory above the 16-megabyte line. In z/VM, it represents total virtual memory.

The default value is 16,384 K (16 M). The minimum value is 1,024 K.

The minimum value for POOLMEMORY is 1,024 K. You can set a maximum threshold for POOLMEMORY when you install Pooled Tables.

- ❑ In z/OS, POOLMEMORY represents memory above the 16-megabyte line. You can also control the total amount of memory available from the operating system above the 16 megabyte line by coding REGION=*n*M on your JCL job card, where *n* is greater than 16.
- ❑ In z/VM, POOLMEMORY represents total virtual memory.

You can also set POOLMEMORY from the command line, during FOCUS initialization (in the PROFILE FOCEXEC), or within an application.

When POOLMEMORY is insufficient to execute every request in a cluster simultaneously, Pooled Tables executes them in iterations, producing as many reports as it can in memory in the first iteration and staging data for the remaining reports in a FOCPOOLT work file it creates for this use. The remaining reports are produced from FOCPOOLT in subsequent iterations, so the original data source is still only accessed once at the outset. When a cluster can be produced directly from memory, no FOCPOOLT file is created.

Syntax: How to Reserve Memory for Other Modules

```
SET POOLRESERVE =n
```

where:

n

Is an amount of memory (kilobytes) reserved for other modules that Pooled Tables cannot use.

In z/VM, the default is 1,024K. In z/OS, it is 100K.

POOLRESERVE reserves memory for use by other modules during the Pooled Tables request-parsing and decision-making processes. For example, initial access to SQL/DS requires loading of Information Builders interface code and IBM modules (this memory will not be made available to Pooled Tables).

You can change either at installation time, by setting POOLRESERVE from the command line during FOCUS initialization (in the PROFILE FOCEXEC), or within your application.

Suggested values for POOLRESERVE are:

```
Running an interface (not in saved segment): 1024 K
Running an interface (in saved segment):      256 K
Using SyncSort as the external sort:          512 K
Using any other sort:                         128 K
```

Memory for such activities is not used in the Pooled Tables case until the common read is executed. After loading these modules, you can reduce POOLRESERVE, perhaps to zero. If the Information Builders interface and IBM load modules are stored in a saved segment, you can even reduce POOLRESERVE before executing Pooled Tables.

The Pooled Tables Trace facility (see [Observing the Results of Pooling \(TRACEON\)](#) on page 932) displays actual memory allocations for each report and the statistics used to calculate it.

Example: Using a Temporary FOCPOOLT Work File

If a cluster contained 30 report requests, each requiring 1 megabyte of memory, and 10 megabytes was all the memory allocated for POOLMEMORY, Pooled Tables would retrieve data for all 30 reports but produce only the first 10 reports directly from memory (the first iteration), writing the records for the remaining 20 reports to the FOCPOOLT work file. In the next iteration, Pooled Tables would read data for the next 10 reports from FOCPOOLT and process those. In a third iteration it would process the data for the final 10 reports.

FOCPOOLT retrievals are more efficient than going back to the data source, because the data is pre-screened and formatted, and because Pooled Tables collected accurate record counts (ESTRECORDS) when it wrote records for the second and subsequent iterations to FOCPOOLT. With accurate memory requirements calculated, Pooled Tables performance is optimized.

Sharing Selection Criteria and Filters Across Requests in a Pool

Selection statements that appear in every report request in a cluster are automatically applied just once during Pooled Tables retrievals. To qualify, such tests must refer to the same field and apply an equality test (EQ or IS); however the actual values selected need not be the same. For example, if the first report tests WHERE FISCAL_YEAR EQ 1997 and the second tests for WHERE FISCAL_YEAR EQ 1998, Pooled Tables applies the test WHERE FISCAL_YEAR EQ 1997 OR 1998 during data retrieval. Common selection tests greatly reduce the size of answer sets returned.

Pooled Tables can also evaluate common selection criteria not based on equality tests through the FOCUS Filters feature. Filters permit specification of simple or complex selection tests against a common file for all reports. If, for example, all reports in a cluster use WHERE DELETE_FLAG NE 'Y', you can create a filter with that test. Alternately, you could change the test to read WHERE DELETE_FLAG EQ 'N' so that the common selection command is used in the Pooled Tables common read.

Criteria When Pooling Non-Relational Database Requests

Reports against non-relational databases, such as VSAM, IMS, IDMS, FOCUS, and sequential files, must meet several simple criteria to be pooled into one cluster. To qualify, all reports must access the same data source, use the same Master File and share the same access method. All reports in a cluster must also share the same entry point (the reporting view must be from the same segment and, in the case of indexed access, from the same field). Reports against sequential files always meet these criteria and always pool. Reports against joined structures are pooled if they share the same access method to the host file.

Criteria When Pooling Relational Database Requests

Reports against relational databases, such as UDB (DB2) and SQL/DS, can be pooled into the same cluster when they share several common attributes. Like non-relational files, all reports must access the same Master File from the same entry point. Reports requiring SQL aggregation (the generated SQL statements contain the GROUP BY phrase) are not pooled, which assures that the set presented to each report in the pool is accurate. Further, requests against a multi-table relational view must all reference the same tables to be pooled into the same cluster.

If a view contains table A and table B, all reports that reference only fields in table A can be pooled, all reports that reference only fields in table B can be pooled, and all reports referencing fields in both table A and B can be pooled. However, none of the reports in those sets could be pooled with reports from the other sets. This limitation insures that the RDBMS retrieval engine uses the same optimization logic for each report in the set.

Less stringent pooling requirements apply with optimization off (SQL SET OPTIMIZATION OFF). Since FOCUS manages the retrieval and aggregation operations in this case, pooling conditions are the same with optimization off as with non-relational databases. Restrictions regarding common accessed tables and SQL aggregation do not apply.

Pooling benefits obtained with optimization off, versus those gained by allowing the RDBMS to optimize retrievals, vary from case to case. For example, a request requiring an area sweep that returns a large answer set (even with optimization), would be a good candidate to pool with other requests if optimization were turned off.

When the interface trace facility is used for a relational database, the SQL generated for each request is echoed. The SQL is generated during the Pooled Tables parsing phase but is not submitted to the RDBMS. Instead, Pooled Tables constructs an internal request to retrieve all data for the cluster. The SQL SELECT statements generated for the cluster are echoed in the trace, and these are the statements passed to the RDBMS.

SQL SELECT statements generated by Pooled Tables are optimized by the RDBMS. Therefore, the best optimization occurs when all requests in a cluster contain the same equality screening conditions or Filters. In such cases, the screening tests are included in the SQL and passed to the RDBMS for optimization. Without application of common selections or Filters, it is possible that efficiencies gained through RDBMS optimization could be lost in pooling individual requests. Consider these two requests: the first returns a small answer set based on a selection against a key field named KEY1. The second returns a small answer set based on a selection against a key field named KEY2. The independent screening conditions are not included in the SQL generated by Pooled Tables, resulting in an area sweep and a large answer set for the cluster. If the two tests were included as an OR condition in a Filter, the screening operation would be passed to the RDBMS and a much smaller answer set returned to Pooled Tables.

Criteria When Pooling Batch Requests

How to:

Control Automatic Application of Pooling for Batch Processing

Pooled Tables automatically pools batch requests wherever possible if the POOLBATCH SET command is issued in a user's PROFILE or in FOCPARM. A batch is any non-interactive session. In z/OS, this is whenever ddname SYSIN is allocated to a data set. In z/VM, non-interactive jobs occur when ddname SYSIN is defined (FILEDEF) to a file, FOCUS is invoked with the syntax FOCUS IN fileid, or the z/VM session is running disconnected.

Syntax: How to Control Automatic Application of Pooling for Batch Processing

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

where:

OFF

Disables automatic use of Pooled Tables for batch processing. This is the default.

POOLBATCH can be included in the FOCPARM ERRORS, FOCUS PROFILE, a FOCEXEC, or issued in the SYSIN input stream.

SET POOLBATCH=ON has the effect of automatically setting POOL=ON for batch execution. SET POOLBATCH=OFF does not reverse this setting. To disable pooling when POOLBATCH=ON, issue the command SET POOL=OFF.

ON

Enables automatic use of Pooled Tables for batch processing.

Selecting a Sort Utility

How to:

Specify a Sort Utility for Use With Pooled Tables

Limit the Number of Concurrent External Sorts That Can Run

Pooled Tables chooses an in-memory FOCUS sort or an external sort based on report-size estimates. Normally, the FOCUS sort is used for reports under a megabyte, and external sorts in other cases. The limiting factor on concurrently executing sorts is the amount of memory available to Pooled Tables. While Pooled Tables can execute up to 26 external sorts, this is controlled by the MAXEXTSRSTS setting and by how much memory is provided below the 16-megabyte line in z/OS. In z/VM, only one external sort can be executed with SyncSort. When it is practical, the FOCUS sort is substituted for the external sort when external sorts are limited but memory is available.

Syntax: How to Specify a Sort Utility for Use With Pooled Tables

```
SET SORTLIB = sorttype
```

where:

sorttype

Can be one of the following:

SYNCSORT identifies the external sort utility as SYNCSORT.

DFSORT identifies the external sort utility as DFSORT.

VMSORT identifies the external sort utility as VMSORT.

MVSMGSS identifies the external sort utility as SYNCSORT and its messages are displayed (z/OS only).

MVSMGDF identifies the external sort utility as DFSORT and its messages are displayed (z/OS only).

Syntax: How to Limit the Number of Concurrent External Sorts That Can Run

```
SET MAXEXTSRSTS=n
```

where:

n

Is the maximum number (from 1 to 26) of concurrent external sorts permitted. The default is 26.

In z/VM, only one version of SyncSort can run concurrently. If you use SyncSort in z/VM, the value of MAXEXTSRTS is assumed to be 1.

Observing the Results of Pooling (TRACEON)

How to:

Turn on the Pooled Tables Trace

Turn Off the Trace Facility

Reference:

Trace Output

The Pooled Tables trace facility breaks down pools into sub-pools and clusters, warns when memory allocation is insufficient, and displays report statistics. The trace facility shows how pools were executed to help developers tune their applications.

Syntax: How to Turn on the Pooled Tables Trace

```
SET TRACEUSER=ON
SET TRACEOFF=ALL
SET TRACEON=POOLTABL // {CLIENT | FSTRACE }
```

where:

CLIENT

Directs trace output to the terminal.

FSTRACE

Is a ddname where trace output can be directed. You must allocate a FILEDEF ddname FSTRACE to a sequential data source. Recommended DCB attributes are RECFM=F and LRECL=160.

Note: SET TRACEUSER=ON is required to enable the trace facility. SET TRACEOFF=ALL ensures that no traces are activated. When you then activate the Pooled Tables trace, it will be the only trace activated.

Syntax: How to Turn Off the Trace Facility

```
SET TRACEOFF=POOLTABL
```

where:

POOLTABL

Ends the Pooled Tables Trace facility.

Reference: Trace Output

These messages indicate sub-pool boundary encounters:

```
Sub pool boundary--prior output required as input
Sub pool boundary--FOCUS/SET command
Sub pool boundary--DEFINE ADD
Sub pool boundary--new MASTER name
Sub pool boundary--new DEFINE clears pre-pool DEFINE
This command will run now, outside of pooling:
A DEFINE ADD will run now, outside of pooling.
```

These messages indicate cluster boundary encounters:

```
Cluster boundary--new master name
Cluster boundary--single-table cluster
Cluster boundary--new alternate view
Cluster boundary--new pool flag
Cluster boundary--new pool condition
Cluster boundary--mid-stream DEFINE
Cluster boundary--new entry segment
Cluster boundary--too many verb objects
```

These messages indicate reports that cannot be pooled (single-table clusters):

```
Single-table cluster--REDEFINED real field
Single-table cluster--User subroutine not known safe
Single-table cluster--self-referential DBA/filter
Single-table cluster--INCLUDES/EXCLUDES selection
Single-table cluster--too many test literals
Single-table cluster--complex test on index
Single-table cluster--$ORTPARM allocated
Single-table cluster--REDEFINED constant real field
Single-table cluster--RANKED BY
Single-table cluster--COUNT DISTINCT
Single-table cluster--RECAP
Single-table cluster--COUNT is a verb object
Single-table cluster--indexed view via AUTOINDEX
Single-table cluster--EMR
Single-table cluster--ON TABLE SET
Single-table cluster--TEXT field
Single-table cluster--PREVIEW mode
Single-table cluster--ALL = ON/PASS
Single-table cluster--per message above
Single-table cluster--indexed view for FOCUS database
Single-table cluster--non-poolable interface request
Single-table cluster--too many verb objects
```

These trace messages appear during the creation and execution of clusters and iterations:

```
Building cluster x...
Cluster contains n table(s)
Cluster n dedicated to command x
Clusters built; sub pool contains x cluster(s).
***** Stack before 1st cluster: *****
***** Stack before nth cluster: *****
***** Begin union table *****
**** Stack before nth iteration: ****
```

During the parsing phase of Pooled Tables, the following statistics are displayed for each report. These indicate whether a report request can be pooled and under what conditions. All reports with the same pooling criteria can be pooled together.

```
Entry Segment      : x
Relational Flag    : y
Pool Flag          : z
Condition Length   : n
Condition          : c
```

After execution of a pooled report, the output from ? STAT is included in the trace. The entries for TRACKIO and MINIO are included in the output, but their values are not populated. In addition, the following statistics are included:

```
TRAVERSAL MTHD =          x          ENTRY SEGMENT =          I
FOCUS SORT MEM =          y1         EXTSORT MEMORY =          y2
ALGORITHM USED =          z
```

The following trace messages indicate limitations imposed on Pooled Tables by users in executing reports under less than the most favorable conditions, based on parameters provided for POOLMEMORY, POOLRESERVE, ESTRECORDS, and ESTLINES or available memory. These messages do not inhibit the execution of Pooled Tables, but make it less efficient. To correct these situations, replace the values for ESTRECORDS and ESTLINES with accurate values or increase the memory allocated for Pooled Tables.

```
# concurrent external sorts reduced from x to y by below-16M shortage
Minimum sort memory forces iterations
Warning--POOLMEMORY desired = x but only y is available
Warning: actual line count (x) exceeds lines estimate (y) in heavy
aggregation case
Warning: records estimate (x) off by more than 10%-actual record count=y
Warning: lines estimate (x) off by more than 10%-actual line count = y
```

Installing the Pooled Tables Option

How to:

- Install on All Systems
- Install on z/OS
- Install on z/VM
- Configure Pooled Tables

This section provides installation instructions for all systems: IMS, z/OS, and z/VM.

Procedure: How to Install on All Systems

Enable Pooled Tables for your release of FOCUS by including the following command in FOCPARM:

```
SET POOLFEATURE = ON
```

To disable Pooled Tables, include the following command in the FOCPARM file:

```
SET POOLFEATURE = OFF
```

If FOCPARM does not contain a SET POOLFEATURE command, FOCUS assumes Pooled Tables is disabled.

The maximum memory above 16 megabytes that can be requested with the SET POOLMEMORY command can be restricted by including the SET MAXPOOLMEM = *n* command in FOCPARM.

To make POOL = ON the default for all batch jobs, include the command SET POOLBATCH = ON. This must follow the SET POOLFEATURE = ON command in FOCPARM.

Each of the commands is also included in member FOCPARM of ERRORS.DATA (z/OS) or in the file FOCPARM ERRORS (CMS).

Procedure: How to Install on z/OS

Include the POOLFEATURE, POOLBATCH, and MAXPOOLMEM commands in member FOCPARM in ERRORS.DATA as outlined above. Refer to your FOCUS documentation to change the default allocation for the file FOCPOOLT.

If you use DFSort and try to run more than 10 sorts concurrently, DFSort displays this message:

```
ICE149A      DFSORT IS NOT LICENSED FOR USE ON THIS SYSTEM. RETURN CODE 12,
             REASON CODE 4.
```

This causes FOCUS to ABEND. Issue the command SET MAXEXTSRSTS=10 to avoid this symptom temporarily. IBM has fixed this problem with APAR OW29152. Order IBM PTF UW41671 if you run SMS Release 1.3. Order IBM PTF UW41672 if you run SMS 1.4.

Procedure: How to Install on z/VM

Include the POOLFEATURE, POOLBATCH, and MAXPOOLMEM commands in the file FOCPARM ERRORS as outlined above. Change the value of POOLRESERVE in FOCPARM ERRORS if appropriate for your installation.

Syntax: How to Configure Pooled Tables

To configure Pooled Tables, include the following commands in the FOCPARM file

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

where:

OFF

Disables Pooled Tables for this FOCUS site. OFF is the default value.

ON

Enables Pooled Tables for this FOCUS site.

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

where:

OFF

Does not enable automatic use of Pooled Tables for batch processing. This is the default.

POOLBATCH can be included in the FOCPARM ERRORS, FOCUS PROFILE, a FOCEXEC, or issued in the SYSIN input stream.

SET POOLBATCH=ON has the effect of automatically setting POOL=ON for batch execution.

SET POOLBATCH=OFF will not reverse this setting. To disable pooling when POOLBATCH=ON, issue the command SET POOL=OFF.

ON

Enables automatic use of Pooled Tables for batch processing.

```
SET MAXPOOLMEM = n
```

where:

n

Sets an upper limit in kilobytes for memory above 16 megabytes that users can allocate in the SET POOLMEMORY command. The default is 32,768 K (32 M) and the minimum is 1,024K.

19 | Creating Financial Reports With Financial Modeling Language (FML)

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 a function of the data.

Topics:

- ❑ Reporting With FML
- ❑ Creating Rows From Data
- ❑ Supplying Data Directly in a Request
- ❑ Performing Inter-Row Calculations
- ❑ Referring to Rows in Calculations
- ❑ Referring to Columns in Calculations
- ❑ Referring to Cells in Calculations
- ❑ Using Functions in RECAP Calculations
- ❑ Inserting Rows of Free Text
- ❑ Adding a Column to an FML Report
- ❑ Creating a Recursive Model
- ❑ Reporting Dynamically From a Hierarchy
- ❑ Customizing a Row Title
- ❑ Formatting an FML Report
- ❑ Suppressing the Display of Rows
- ❑ Saving and Retrieving Intermediate Report Results
- ❑ Creating HOLD Files From FML Reports

Reporting With FML

FML is an integrated extension of the TABLE command. By adding the FOR phrase and the RECAP command, you can handle an expanded range of applications.

Note: MORE is not supported in FML requests.

In conjunction with Dialogue Manager, FML can be used to evaluate "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, they can easily be changed. FML includes the following facilities:

- ❑ Row/column formatting: You can specify results in a row-by-row, column-by-column fashion (see [Performing Inter-Row Calculations](#) on page 953).
- ❑ 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 1002).
- ❑ 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 a Request](#) on page 952).
- ❑ Recursive reporting: 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 can use recursive reports to produce a cash flow projection (see [Creating a Recursive Model](#) on page 973).
- ❑ Dynamic reporting from a chart of accounts or a similar hierarchy of information. You can create a report that changes as the organization of information changes, ensuring that you automatically retrieve information that reflects the latest structure and its values. There is no need to alter either the Master File or the report request. See [Reporting Dynamically From a Hierarchy](#) on page 975.

Example: Sample FML Request

This example produces a simple asset sheet, contrasting the results of two years. It illustrates many key features of the Financial Modeling Language (FML). Numbers to the left of the procedure lines correspond to explanations that follow the request.

```

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"
   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
   "</2 *** PRELIMINARY ASSET SHEET BASED ON UNAUDITED FIGURES ***"
END

```

- 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 in the FINANCE data source. 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 in the report.
5. Free text can be incorporated at any point in an FML report, similar to underlines.
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:

PAGE 1

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
*** PRELIMINARY ASSET SHEET BASED ON UNAUDITED FIGURES ***		

PAGE 2

COMPARATIVE ASSET SHEET		
	YEAR	
	1983	1982
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

In this section:

- Creating Rows From Multiple Records
- Using the BY Phrase in FML Requests
- Combining BY and FOR Phrases in an FML Request

How to:

- Retrieve FOR Field Values From a Data Source

A normal TABLE request sorts rows of a report according to the BY phrase you use. The data retrieved is sorted from either low-to-high or high-to-low, as requested. The rows may be limited by a screening phrase to a specific subset, but:

- ❑ They appear in a sort order.
- ❑ Rows appear only for values that are retrieved from the file.
- ❑ You can only insert free text between 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 creates a matrix in which you can structure your report row-by-row. This organization gives you greater control over the data that is incorporated into a report, and 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 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, addresses, and values so that you can point to the individual cells formed at each intersection (as on a spreadsheet).

Syntax: **How to Retrieve FOR Field Values From a Data Source**

The syntax for specifying rows is:

```
FOR fieldname [NOPRINT]
value [OR value OR...] [AS 'text'] [LABEL label] OVER
.
.
[value [OR value ...] [AS 'text'] [LABEL label]]
END
```

where:

fieldname

Is a field name in the data source.

value

Is the value (also known as a tag value) describing the data that is retrieved for this row of the report.

AS 'text'

Enables you to assign a name to a tag value, which replaces the tag value in the output. Enclose the text in single quotation marks.

label

Assigns a label to the row for reference in a RECAP expression. The label can be up to 66 characters and cannot have blanks or special characters. Each explicit label you assign must be unique.

Even if you assign an explicit label, the positional label (R1, R2, etc.) is retained internally.

By default, a tag value for a FOR field (like 1010) may be added only once to the FML matrix. However, if you wish to add the same value of a FOR field to the matrix more than once, you can turn on the FORMULTIPLE parameter (the default setting is OFF). See [How to Use the Same FOR Field Value in Multiple Rows](#) on page 948.

See the Using Functions manual for information about the FMLFOR, FMLLIST, and FMLINFO functions that return the tag values used in an FML request.

Example: Creating Rows From Values in a Data Source

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

Creating Rows From Multiple Records

How to:

Sum Values in Rows With the OR Phrase

Identify a Range of Values With the TO Phrase

Use Masking Characters to Retrieve Tag Values

Use the Same FOR Field Value in Multiple Rows

There are different ways to combine multiple values 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. See [How to Sum Values in Rows With the OR Phrase](#) on page 945.
- ❑ The TO phrase to identify a range of tag values on which to report. See [How to Identify a Range of Values With the TO Phrase](#) on page 946.
- ❑ A mask to specify a group of tag values without having to name each one. See [How to Use Masking Characters to Retrieve Tag Values](#) on page 947.

By default, a FOR field value can only be included in a single row of an FML matrix. However, by turning on the FORMULTIPLE parameter, you can include the same data value in multiple rows in the FML matrix. For example, the same value can exist as a solitary value in one row, be part of a range in another row, and be used in a calculation in a third row. See [How to Use the Same FOR Field Value in Multiple Rows](#) on page 948.

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
value1 OR value2 [OR valuen...] [AS 'text'] [LABEL label] [OVER]
.
.
.
```

where:

fieldname

Is a field name in the data source.

value1, value2, valuen

Are the tag values to be retrieved and summed.

AS '*text*'

Assigns a title to the combined tag values. Enclose the text in single quotation marks.

label

Assigns a label to the row for reference in a RECAP expression. The label can be up to 66 characters and cannot have blanks or special characters. Each explicit label you assign must be unique.

Even if you assign an explicit label, the positional label (R1, R2, etc,) is retained internally.

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**

To sum the values of a range of tags in a single report row, use the TO phrase in the FOR phrase. The syntax is:

```
FOR fieldname  
value1 TO value2 [AS 'text'] [LABEL label] [OVER]
```

where:

fieldname

Is a field name in the data source.

value1

Is the tag value at the lower limit of the range.

TO

Is the required phrase.

value2

Is the tag value at the upper limit of the range.

AS '*text*'

Assigns a title to the combined tag values. Enclose the text in single quotation marks.

label

Assigns a label to the row for reference in a RECAP expression. The label can be up to 66 characters and cannot have blanks or special characters. Each explicit label you assign must be unique.

Even if you assign an explicit label, the positional label (R1, R2, etc.) is retained internally.

Example: Identifying a Range of Values

Since CASH accounts in the LEDGER system are identified by the tags 1010, 1020, and 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 Characters to Retrieve Tag Values

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,

```
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 Characters 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\$\$, are 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
```

The output is:

	AMOUNT

CASH	21,239
ACCOUNTS RECEIVABLE	18,829
INVENTORY	27,307

Syntax: **How to Use the Same FOR Field Value in Multiple Rows**

You can use the same value of a FOR field in many separate rows (whether alone, as part of a range, or in a calculation) by including the following syntax before or within an FML request:

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

or

```
ON TABLE SET FORMULTIPLE {ON|OFF}
```

where:

ON

Enables you to reference the same value of a FOR field in more than one row in an FML request.

With FORMULTIPLE set to ON, a value retrieved from the data source is included on every line in the report output for which it matches the tag references.

OFF

Does not enable you to include the same value in multiple rows. OFF is the default value.

With FORMULTIPLE set to OFF, multiple tags referenced in any of these ways (OR, TO, *) are evaluated first for an exact reference or for the end points of a range, then 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 is 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.)

See [Reporting Dynamically From a Hierarchy](#) on page 975.

Example: Referencing the Same Value in More Than One Row

This request retrieves the tag values for accounts 1010, 1020, and 1030, and lists corresponding values individually. It then aggregates the same values and displays the sum as TOTAL CASH. Similarly, the tag values for accounts 1100 and 1200 are displayed as detail items, and then summarized as TOTAL NON-CASH ASSETS.

```

SET FORMULTIPLE = ON
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
1010 OR 1020 OR 1030 AS 'TOTAL CASH'  OVER
" "                               OVER
1100 AS 'ACCOUNTS RECEIVABLE'   OVER
1200 AS 'INVENTORY'             OVER
BAR                               OVER
1100 TO 1200 AS 'TOTAL NON-CASH ASSETS'
END

```

The output is:

	AMOUNT

CASH ON HAND	8,784
DEMAND DEPOSITS	4,494
TIME DEPOSITS	7,961

TOTAL CASH	21,239
ACCOUNTS RECEIVABLE	18,829
INVENTORY	27,307

TOTAL NON-CASH ASSETS	46,136

Example: Using Tags From External Files

In this example, the values 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 tag values 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. Note that the BY ROWS OVER phrase is not supported in a request that uses the FOR phrase.

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
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

```

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 computes a non-zero value only for the EAST region:

```
RECAP X=IF REGION EQ 'EAST' THEN .25*CASH ELSE 0;  
AS 'AVAILABLE FOR DIVIDENDS'
```

See [Performing Inter-Row Calculations](#) on page 953.

Supplying Data Directly in a Request

How to:

Supply Data Directly in a Request

In certain cases, you may need to include additional constants (such as 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] OVER
```

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'

Enables you to assign a title to the data row. Enclose the text in single quotation marks. Without this entry, the row title is blank on the report.

label

Assigns a name to the data row 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 (.87 and 1.67) are provided for the exchange rates of euros and pounds, respectively:

```
DEFINE FILE LEDGER
EUROS/I5C=AMOUNT;
POUNDS/I5C=3.2*AMOUNT;
END

TABLE FILE LEDGER
SUM EUROS AS 'EUROPE,DIVISION'
POUNDS AS 'ENGLISH,DIVISION'
FOR ACCOUNT
1010 AS 'CASH--LOCAL CURRENCY' LABEL CASH          OVER
DATA .87 , 1.67 , $ 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:

	EUROPE DIVISION	ENGLISH DIVISION
	-----	-----
CASH--LOCAL CURRENCY	8,784	28,106
EXCHANGE RATE	.87	1.67
US_DOLLARS	7,642	46,937

Performing Inter-Row Calculations**How to:**

Define Inter-Row Calculations

Reference:

Usage Notes for RECAP

The RECAP command enables 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 value that results from 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. See [Referring to Rows in Calculations](#) on page 955.

Syntax: **How to Define Inter-Row Calculations**

```
RECAP calcname [/format]=expression; [AS 'text']
```

where:

RECAP

Is the required command name. It should begin on a line by itself.

calcname

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 name also serves as an explicit label. See [Referring to Rows in Calculations](#) on page 955.

format

Is the USAGE format of the calculated value. It cannot exceed the column width. The default is the format of the column in which the calculated value is displayed.

expression

Can be any calculation available with the DEFINE command (including IF... THEN ... ELSE syntax, functions, excluding DECODE and 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. See [Using Functions in RECAP Calculations](#) on page 966 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. See [Referring to Rows in Calculations](#) on page 955, [Referring to Columns in Calculations](#) on page 958, and [Referring to Cells in Calculations](#) on page 965.

AS '*text*'

Changes the default title of the row. By default, the name of the RECAP value is displayed as the row title in output. The AS phrase replaces the default. Enclose the text in single quotation marks.

Reference: Usage Notes for RECAP

- ❑ RECAP expressions refer to other rows in the model by 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. This feature can be used 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 in Calculations**How to:**

Assign an Explicit Row Label

FML assigns a default positional label to each TAG, 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 need not 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 name of the calculated value on the left of the equal sign can be used as a label.

In addition to their role in RECAP calculations, you can use labels to format rows in an FML report. See [Formatting an FML Report](#) on page 992.

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'

Assigns a different name to the row for the report. Enclose the text in single quotation marks.

label

Assigns a label to a row for reference in a RECAP expression or a StyleSheet declaration. The label can be up to 66 characters and cannot have blanks or special characters. Each explicit label you assign must be unique.

Even if you assign an explicit label, the positional label (R1, R2, etc.) is retained internally.

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 CA, 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 CA      OVER
1100 AS 'ACCOUNTS RECEIVABLE' LABEL AR      OVER
1200 AS 'INVENTORY'          LABEL INV    OVER
BAR
RECAP CURASST/I5C= CA + AR + INV;
END
```

The output is:

	AMOUNT

CASH	21,239
ACCOUNTS RECEIVABLE	18,829
INVENTORY	27,307

CURASST	67,375

Note that the RECAP value could subsequently be referred to by the name CURASST, which functions as an explicit label.

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 statement 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.

Tip: If you set the FORMULTIPLE parameter ON, you can repeat the row without giving it another name. See [Creating Rows From Multiple Records](#) on page 944.

Referring to Columns in Calculations

In this section:

- Referring to Column Numbers in Calculations
- Referring to Contiguous Columns in Calculations
- Referring to Column Addresses in Calculations
- Referring to Relative Column Addresses in Calculations
- Applying Relative Column Addressing in a RECAP Expression
- Controlling the Creation of Column Reference Numbers
- Referring to Column Values in Calculations

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 notation to control the creation of column reference numbers.
- ❑ Column values.

Referring to Column Numbers in Calculations

A calculation may be performed for one column or for a specific set of columns. To identify the columns, place the column number in parentheses after the label name.

Example: Referring to Column Numbers in a RECAP Expression

```
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 value is only calculated for the column specified.

The output is:

	CURRENT YEAR	LAST 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 in Calculations

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 in Calculations

How to:

Use Column Addressing in a RECAP Expression

When you need a calculation for every other or every third column instead of every column, you can supply a factor, or column address, to do this. Column addressing is useful when several data fields are displayed within each value of a column sort.

Syntax: How to Use Column Addressing in a RECAP Expression

The left-hand side of the expression has the form:

```
value(s,e,i)[/format]=
```

where:

value

Is the name you assign to the result of the RECAP calculation.

s

Is the starting column.

e

Is the ending column (it may be * to denote all columns).

I

Is the increment factor.

format

Is the USAGE format of the calculated value. The default value is the format of the original column.

Example: Applying Column Addressing in a RECAP Expression

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, control the placement of the results with a RECAP in the form:

```
RECAP calcname(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 in Calculations

A calculation can use a specific column as a base, and refer to all other columns by their displacement from that column. The column to the left of the base column has a displacement of -1 relative to the base column. The column to the right has a displacement of +1. 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, the column must already have been retrieved, or its value is zero.

Applying Relative Column Addressing in a RECAP Expression

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'
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	

Controlling the Creation of Column Reference Numbers

How to:

Control the Creation of Column Reference Numbers

Column notation assigns a sequential column number to each column in the internal matrix created for a report request. If you want to control the creation of column reference numbers for the columns that are used in your report, use the CNOTATION column notation command.

Because column numbers refer to columns in the internal matrix, they are assigned after retrieval and aggregation of data are completed. Columns created and displayed in a report are stored in the internal matrix, and columns that are not displayed in a report may also be generated and stored in the internal matrix. Columns stored in the internal matrix include calculated values, reformatted field values, BY fields, fields with the NOPRINT option, and certain RECAP calculations such as FORECAST and REGRESS. Every other column in the internal matrix is assigned a column number by default which means you have to account for all internally generated columns if you want to refer to the appropriate column value in your request.

You can change the default assignment of column reference numbers by using the SET CNOTATION=PRINTONLY command which assigns column numbers only to columns that display in the report output. You can use column notation in COMPUTE and RECAP commands to refer to these columns in your request.

Syntax: How to Control the Creation of Column Reference Numbers

```
SET CNOTATION={ALL | PRINTONLY | EXPLICIT}
```

where:

ALL

Assigns column reference numbers to every column in the internal matrix. ALL is the default value.

PRINTONLY

Assigns column reference numbers only to columns that display in the report output.

EXPLICIT

Assigns column reference numbers to all fields referenced in the request, whether displayed or not.

Note: CNOTATION is not supported in an ON TABLE phrase.

Referring to Column Values in Calculations

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 addressed by this value in a RECAP calculation.

Example: Referring to a Column by Its Value in a RECAP Expression

The following request uses a factor that depends on the value of the ACROSS field (YEAR) to calculate the inventory cost for each year. It then calculates the profit by summing the assets and subtracting the inventory cost for each year.

```
TABLE FILE LEDGER
SUM AMOUNT ACROSS YEAR
FOR ACCOUNT
10$$ AS 'CASH' LABEL CASH OVER
1100 AS 'ACCOUNTS RECEIVABLE' LABEL RECEIVE OVER
BAR OVER
1200 AS 'INVENTORY VALUE' LABEL INVENT OVER
RECAP INVENTORY_FACTOR/F5.2 = IF YEAR LT '1986'
    THEN 1.1 ELSE 1.25; AS 'INVENTORY COST FACTOR' OVER
RECAP INVENTORY_COST = INVENTORY_FACTOR * INVENT;
    AS 'INVENTORY COST' OVER
BAR OVER
RECAP PROFIT = CASH + RECEIVE - INVENTORY_COST;
END
```

The output is:

	YEAR		
	1985	1986	1987
CASH	5,663	7,001	8,575
ACCOUNTS RECEIVABLE	5,295	6,250	7,284
INVENTORY VALUE	7,754	9,076	10,477
INVENTORY COST FACTOR	1.10	1.25	1.25
INVENTORY COST	8,529	11,345	13,096
PROFIT	2,429	1,906	2,763

Referring to Cells in Calculations

How to:

Use Cell Notation for Rows and Columns in a RECAP Expression

You can refer to columns and rows using a form of cell notation that identifies the intersection of a row and a column as $E(r, c)$.

Syntax: How to Use Cell Notation for Rows and Columns in a RECAP Expression

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 a RECAP Expression

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 Functions in RECAP Calculations

How to:

Call a Function in a RECAP Command

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 functions 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 functions.

Syntax: How to Call a Function in a RECAP Command

```
RECAP calcname[(s,e,i)][/format]=function
(input1,...,inputn,'format2');
```

where:

calcname

Is the name you assign to the calculated value.

(*s,e,i*)

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 format. If the calculation format is larger than the column width, the value displays in that column as asterisks.

function

Is the name of the function, up to eight characters long. 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 function. They may include numeric constants, alphanumeric literals, row and column references ® notation, E notation, or labels), or names of other RECAP calculations.

Make sure that the values being passed to the function agree in number and type with the arguments as coded in the function.

format2

Is the format of the return value, which must be enclosed in single quotation marks.

Example: Calling a Function in a RECAP Command

Suppose you have a function named INVEST in your private collection of functions (INVEST is not available in the supplied library), and it calculates an 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 function.

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 INVAIL = INVEST(CASH,TOTCAS,THISDATE,'D12.2'); AS
'AVAIL. FOR INVESTMENT' OVER
BAR AS '='
END
```

The output is:

```
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
=====
```

Inserting Rows of Free Text

How to:

Insert Data Variables in Text Rows

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'
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. (In a RECAP, the calculated value functions as the 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.

Example: **Inserting a Data Variable in a Text Row**

In this example, the RECAP value CURASST is suppressed by the NOPRINT command, and inserted instead as a data variable in the text row.

```
SET PAGE-NUM=OFF
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
10$$ AS 'Cash'                LABEL CA    OVER
1100 AS 'Accounts Receivable' LABEL AR    OVER
1200 AS 'Inventory'          LABEL INV   OVER
RECAP CURASST/I5C= CA + AR + INV; NOPRINT OVER
"Current Assets: <CURASST"
END
```

The output is:

	AMOUNT

Cash	21,239
Accounts Receivable	18,829
Inventory	27,307
Current Assets:	67,375

Adding a Column 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 contains two columns: AMOUNT and FORECAST.

Add columns in an FML request, just as 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 information, see [Creating Temporary Fields](#) on page 205.

Example: Adding a Column to an FML Report

This example uses a COMPUTE command to generate the calculated value CHANGE and display it as a new column in the FML report. The following request generates an FML matrix with four rows and three columns of data.

```
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'
COMPUTE CHANGE/I5C = CUR_YR - LAST_YR;
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'
END
```

	CURRENT YEAR	LAST YEAR	CHANGE
	-----	-----	-----
CASH ON HAND	8,784	7,214	1,570
DEMAND DEPOSITS	4,494	3,482	1,012
TIME DEPOSITS	7,961	6,499	1,462
	-----	-----	-----
TOTAL CASH	21,239	17,195	4,044

Note: The designated calculation is performed on each tag or RECAP row of the report. The RECAP rows, however, may change the calculation.

Example: Adding a New Time Period as a Column

The following request adds a future time period to a report:

```

DEFINE FILE LEDGER
CUR_YR/P5C = AMOUNT;
LAST_YR/P5C = .87*AMOUNT - 142;
END

TABLE FILE LEDGER
SUM AMOUNT
ACROSS YEAR AND COMPUTE 1999/P5C = 2.5*AMOUNT;
FOR ACCOUNT
1010 AS 'CASH ON HAND' OVER
1020 AS 'DEMAND DEPOSITS' OVER
1030 AS 'TIME DEPOSITS' OVER
BAR OVER

RECAP TOTCASH/P5C = R1 + R2 + R3; AS 'TOTAL CASH' OVER
RECAP CHANGE(2,*) = TOTCASH(*) - TOTCASH(*-1);
END

```

The output is:

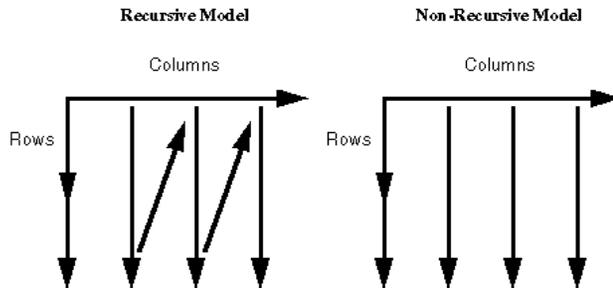
	YEAR			
	1985	1986	1987	1999
CASH ON HAND	1,684	2,100	5,000	4,210
DEMAND DEPOSITS	619	875	3,000	1,548
TIME DEPOSITS	3,360	4,026	575	8,400
TOTAL CASH	5,663	7,001	8,575	14,158
CHANGE		1,338	1,574	5,583

Creating a Recursive Model

Models involving different time periods often require using the ending value of one time period as the starting value for the next. The calculations describing these situations have two characteristics:

- ❑ The labels on one or more RECAP rows are duplicates of other rows. They are used repeatedly to recompute certain 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 (either reuse of labels by different rows, or forward reference to a label in a calculation).

Example: Creating a Recursive Model

The following example illustrates recursive models. Note that one year of ENDCASH becomes the next year of 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:

	LAST_YR	CUR_YR	NEXT_YR
	-----	-----	-----
STARTING CASH	7,903.64	9,024.00	10,374.00
SALES	4,986.00	6,000.00	7,378.20
COST	3,864.15	4,650.00	5,718.10
	-----	-----	-----
PROFIT	1,121	1,350	1,660
ENDCASH	9,024	10,374	12,034

Reporting Dynamically From a Hierarchy

In this section:

- Requirements for FML Hierarchies
- Displaying an FML Hierarchy
- Consolidating an FML Hierarchy
- Loading a Hierarchy Manually

Hierarchical relationships between fields can be defined in a Master File, and automatically displayed using the Financial Modeling Language (FML). The parent and child fields must share data values, and their relationship should be hierarchical. The formats of the parent and child fields must both be either numeric or alphanumeric.

For example, suppose that:

- ❑ Employee and manager IDs are contained within an employee data source.
- or
- ❑ A general ledger data source contains both an account number field and an account parent field.

By examining these fields, it is possible to construct the entire organization chart or chart of accounts structure. However, to print the chart in a traditional FML report, you need to list the employee IDs or account numbers in the request syntax in the order in which they should appear on the report. If an employee or account is added, removed, or transferred, you have to change the report request to reflect this change in organizational structure. For example:

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT CURR_JOBCODE
FOR EMP_ID
999999999 OVER
222222222 OVER
.
.
.
```

In contrast, with FML hierarchies you can define the hierarchical relationship between two fields in the Master File and load this information into memory. The FML request can then dynamically construct the rows that represent this relationship and display them in the report, starting at any point in the hierarchy. In the example shown, EMP_ID is called the hierarchy field.

Requirements for FML Hierarchies

1. In the Master File. Use the PROPERTY=PARENT_OF and REFERENCE=*hierarchyfld* attributes to define the hierarchical relationship between two fields. See the *Describing Data* manual for information.

The hierarchy must be loaded into memory. This loaded hierarchy is called a chart. If the hierarchy is defined in the Master File and referenced by the FML request, it is loaded automatically. If you want to use a hierarchy defined in a Master File that is not either referenced in the FML request or joined to the Master File referenced in the FML request, issue the LOAD CHART command before issuing the FML request.

The number of charts that can be loaded is 16. Charts are automatically unloaded when the session ends.

2. In the FOR phrase of the FML request. Use the GET/WITH CHILDREN or ADD phrase to retrieve the hierarchical data starting at a specific point in the hierarchy.

To use FML hierarchies, the FOR field must either be:

- ❑ The hierarchy field
- or
- ❑ Used as the join field to a unique segment that has the hierarchy field. In this case, the hierarchy field must be the join field. Note that the condition that the join be unique only applies if the hierarchy is defined in the cross-referenced segment.

In other words, the FOR field must be in a parent-child hierarchy, or linked to one. The latter case allows transaction data that contains the hierarchy field to be joined to a separate data source that contains the hierarchy definition.

As with any FML request, 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.

Example: Defining a Hierarchy in a Master File

The CENTGL Master File contains a charts of accounts hierarchy. The field GL_ACCOUNT_PARENT is the parent field in the hierarchy. The field GL_ACCOUNT is the hierarchy field. The field GL_ACCOUNT_CAPTION can be used as the descriptive caption for the hierarchy field:

```
FILE=CENTGL          , SUFFIX=FOC
SEGNAME=ACCOUNTS, SEGTYPE=S01
FIELDNAME=GL_ACCOUNT,          ALIAS=GLACCT,  FORMAT=A7,
                                TITLE='Ledger,Account', FIELDTYPE=I, $
FIELDNAME=GL_ACCOUNT_PARENT,    ALIAS=GLPAR,   FORMAT=A7,
                                TITLE=Parent,
                                PROPERTY=PARENT_OF, REFERENCE=GL_ACCOUNT, $
FIELDNAME=GL_ACCOUNT_TYPE,      ALIAS=GLTYPE,  FORMAT=A1,
                                TITLE=Type, $
FIELDNAME=GL_ROLLUP_OP,         ALIAS=GLROLL,  FORMAT=A1,
                                TITLE=Op, $
FIELDNAME=GL_ACCOUNT_LEVEL,     ALIAS=GLLEVEL, FORMAT=I3,
                                TITLE=Lev, $
FIELDNAME=GL_ACCOUNT_CAPTION,   ALIAS=GLCAP,   FORMAT=A30,
                                TITLE=Caption,
                                PROPERTY=CAPTION, REFERENCE=GL_ACCOUNT, $
FIELDNAME=SYS_ACCOUNT,         ALIAS=ALINE,   FORMAT=A6,
                                TITLE='System,Account,Line', MISSING=ON, $
```

The CENTSYSF data source contains detail-level financial data. This is unconsolidated financial data for a fictional corporation, CenturyCorp. It is designed to be separate from the CENTGL database as if it came from an external accounting system. It uses a different account line system (SYS_ACCOUNT) which can be joined to the SYS_ACCOUNT field in CENTGL. Data uses "natural" signs (expenses are positive, revenue negative).

```
FILE=CENTSYSF        , SUFFIX=FOC
SEGNAME=RAWDATA      , SEGTYPE=S2
FIELDNAME = SYS_ACCOUNT , , A6 , FIELDTYPE=I,
                                TITLE='System,Account,Line', $
FIELDNAME = PERIOD      , , YYM , FIELDTYPE=I, $
FIELDNAME = NAT_AMOUNT  , , D10.0 , TITLE='Month,Actual', $
FIELDNAME = NAT_BUDGET  , , D10.0 , TITLE='Month,Budget', $
FIELDNAME = NAT_YTDAMT  , , D12.0 , TITLE='YTD,Actual', $
```

Displaying an FML Hierarchy

How to:

Display an FML Hierarchy

The GET CHILDREN and WITH CHILDREN commands dynamically retrieve and display hierarchical data on the FML report. GET CHILDREN displays only the children, not the parent value referenced in the command. WITH CHILDREN displays the parent and then the children.

Syntax: How to Display an FML Hierarchy

```
TABLE FILE filename
{PRINT|SUM} ....
FOR hierarchyfld
parentvalue {GET|WITH} CHILD[REN] [n|ALL] [AS CAPTION|'text'] [LABEL label]
.
.
.
END
```

where:

filename

Is the name of the file to be used in the FML request. If the hierarchy for this request cannot be loaded automatically, it must have been loaded previously by issuing the LOAD CHART command.

hierarchyfld

Is the hierarchy field name. If the request references a joined structure, the name must be the field name from the host file. The alias name is not supported.

parentvalue

Is the parent value for which the children are to be retrieved.

GET CHILDREN

Displays the hierarchy starting from the first child of the specified *parentvalue*. It does not include the parent in the display. (This corresponds to the FML syntax CHILD1 OVER CHILD2 OVER...)

WITH CHILDREN

Displays the hierarchy starting from the specified *parentvalue*. It includes the parent in the display. (This corresponds to the FML syntax *parentvalue* OVER CHILD1 OVER CHILD2 OVER ...)

n|ALL

Is a positive integer from 1 to 99, specifying the number of levels of the hierarchy to display. If a number greater than 99 is specified, a warning message is displayed and *n* is set to 99. The default value is 1. Therefore, if *n* is omitted, only direct children are displayed. GET or WITH CHILDREN 2 displays direct children and grandchildren. GET or WITH CHILDREN 99 displays children to 99 levels. ALL is a synonym for 99. Each child instance is printed over the one that follows. Successive levels of the hierarchy field are indented two spaces from the previous level.

CAPTION

Indicates that the caption values to display should be taken from the field defined as the CAPTION in the Master File.

Note that the AS CAPTION phrase is supported for tagged rows, including those that do not use the GET/WITH CHILDREN or ADD syntax. However, the hierarchy must be defined (by specifying the PARENT_OF attribute) in order to load and display the caption values. If the hierarchy is not defined, the AS CAPTION phrase is ignored.

'text'

Is a text string to use as the row title for the hierarchy field values. The CAPTION field defined in the Master File is not used as the caption on the report output.

label

Is an explicit row label. Each generated row is labeled with the specified label text.

Note: The hierarchy is displayed sorted by the parent field and, within parent, sorted by the hierarchy field.

See the Using Functions manual for information about the FMLFOR, FMLLIST, FMLCAP, and FMLINFO functions that return the tag values and captions used in an FML request.

Example: Displaying an FML Hierarchy

The following request displays two levels of account numbers, starting from account 3000:

```
SET BLANKINDENT=ON
TABLE FILE CENTGL
PRINT GL_ACCOUNT_PARENT
FOR GL_ACCOUNT
3000 WITH CHILDREN 2
END
```

The output is:

	Parent

3000	1000
3100	3000
3110	3100
3120	3100
3130	3100
3140	3100
3200	3000
3300	3200
3400	3200
3500	3200
3600	3200
3700	3200
3800	3200
3900	3200

Note that if the request specifies GET CHILDREN instead of WITH CHILDREN, the line for the parent value (3000) does not display on the report output.

Example: Displaying an FML Hierarchy With Captions

The following request displays two levels of a charts of accounts hierarchy, starting with account 1000 (the top of the hierarchy), and displays the caption field values instead of the account numbers:

```
TABLE FILE CENTGL
PRINT  GL_ACCOUNT_PARENT
FOR  GL_ACCOUNT
1000 WITH CHILDREN 2 AS CAPTION
END
```

The output is:

	Parent

Profit Before Tax	
Gross Margin	1000
Sales Revenue	2000
Cost Of Goods Sold	2000
Total Operating Expenses	1000
Selling Expenses	3000
General + Admin Expenses	3000
Total R+D Costs	1000
Salaries	5000
Misc. Equipment	5000

Note that if the request specifies GET CHILDREN instead of WITH CHILDREN, the line for the parent value (1000, Profit Before Tax) does not display on the report output.

Consolidating an FML Hierarchy**How to:**

- Create One Summary Row for an FML Hierarchy
- Consolidate FML Hierarchy Data to Any Level and Depth

The ADD command consolidates multiple levels of the hierarchy on one line of the FML report output. ADD can be used alone or in conjunction with GET CHILDREN or WITH CHILDREN. Note that ADD is designed to work with requests that use the SUM command. It is also designed to be used with detail-level data, not data that is consolidated.

When used alone, ADD aggregates the parent and children on one line of the report output, summing the numeric data values included on the line. This corresponds to the FML syntax *parentvalue* or CHILD1 OR CHILD2 OR ...

When used in conjunction with GET CHILDREN, ADD displays one line for each child of the specified parent value. Each line is a summation of that child and all of its children. You can specify the number of levels of children to display (which determines the number of lines generated on the report output) and the depth of summation under each child. By default, only direct children have a line in the report output, and the summary for each child includes all of its children.

When used in conjunction with WITH CHILDREN, ADD first displays a line in the report output that consists of the summation of the parent value and all of its children. Then it displays additional lines identical to those displayed by GET CHILDREN ADD.

In order to use a data record in more than one line of an FML report (for example, to display both detail and summary lines or to consolidate detail data at multiple levels), the following setting is required:

```
SET FORMULTIPLE=ON
```

Syntax: **How to Create One Summary Row for an FML Hierarchy**

```
TABLE FILE filename
SUM....
FOR hierarchyfld
parentvalue ADD [n|ALL] [AS CAPTION|'text'] [LABEL label]
.
.
.
END
```

where:

filename

Is the name of the file to be used in the FML request. If the hierarchy for this request cannot be loaded automatically, it must have been loaded previously by issuing the LOAD CHART command.

hierarchyfld

Is the hierarchy field name. If the request references a joined structure, the name must be the field name from the host file. The alias name is not supported.

parentvalue

Is the parent value that determines the starting point in the hierarchy for the aggregation.

n|ALL

Is a positive integer from 1 to 99, specifying the number of levels of the hierarchy to aggregate. ALL is the default value. Therefore, if *n* is omitted, all children are included in the sum. If *n* is 1, only direct children are included. If *n* is 2, direct children and grandchildren are included. ADD 99 includes up to 99 levels of children. ALL is a synonym for 99.

ADD

Displays the parent and *n* levels of its children on one row, summing the numeric data values displayed on the row. This corresponds to the FML syntax *parentvalue* or CHILD1 OR CHILD2 OR ...

To display the sum of just the children, you must display the parent row, display the summary row, and use a RECAP to subtract the parent row from the sum. For example:

```
FOR ...
parentvalue                                OVER
parentvalue ADD 1                          OVER
RECAP CHILDSUM = R2-R1;
```

CAPTION

Indicates that the caption of the parent value displays for the total row.

Note that the AS CAPTION phrase is supported for tagged rows, including those that do not use the GET CHILDREN or ADD syntax. However, the hierarchy must be defined (by specifying the PARENT_OF attribute) in order to load and display the caption values. If the hierarchy is not defined, the AS CAPTION phrase is ignored.

'text'

Is a text string to use as the row title for the aggregate row. The CAPTION field defined in the Master File is not used as the caption on the report output.

label

Is an explicit row label. Each generated row is labeled with the specified label text.

Example: Displaying One Summary Line for an FML Hierarchy

The CENTSYSF data source contains detail-level financial data. To use the account hierarchy in the CENTGL data source with this financial data, the two data sources are joined. The data in CENTSYSF is stored with natural signs, which means, in financial terms, that revenues and liabilities are stored as negative numbers. The portion of the hierarchy used in this request contains only positive data.

Note that the join is not required to be unique, because the hierarchy is defined in the host segment.

First the WITH CHILDREN command displays the lines of the hierarchy starting with account 3100 (Selling Expenses). Note that only accounts with no children are populated in this detail-level data source. The ADD command then creates one line that is the sum of account 3100 and all of its children:

```

SET FORMULTIPLE = ON
JOIN SYS_ACCOUNT IN CENTGL TO ALL SYS_ACCOUNT IN CENTSYSF
TABLE FILE CENTGL
SUM NAT_AMOUNT/D10.0 NAT_YTDAMT/D10.0
FOR GL_ACCOUNT
3100 WITH CHILDREN ALL AS CAPTION OVER
BAR                                OVER
3100 ADD AS CAPTION
IF PERIOD EQ '2002/03'
END
    
```

The output is:

	Month Actual -----	YTD Actual -----
Selling Expenses	.	.
Advertising	.	.
TV/Radio	1,049,146.	2,954,342.
Print Media	244,589.	721,448.
Internet Advertising	9,542.	29,578.
Promotional Expenses	53,719.	151,732.
Joint Marketing	97,135.	289,799.
Bonuses/Commissions	100,188.	304,199.
	-----	-----
Selling Expenses	1,554,319.	4,451,098.

Syntax: How to Consolidate FML Hierarchy Data to Any Level and Depth

```
TABLE FILE filename
SUM....
FOR hierarchyfld
parentvalue {GET|WITH} CHILD[REN] [n|ALL] ADD [m|ALL]
  [AS CAPTION|'text'] [LABEL label]
.
.
.
END
```

where:

filename

Is the name of the file to be used in the FML request. If the hierarchy for this request cannot be loaded automatically, it must have been loaded previously by issuing the LOAD CHART command.

hierarchyfld

Is the hierarchy field name. If the request references a joined structure, the name must be the field name from the host file. The alias name is not supported.

parentvalue

Is the parent value that determines the starting point in the hierarchy for the aggregation.

GET|WITH

GET specifies that the first line generated on the report is the consolidated line for the first child of the parent value. WITH specifies that the first line generated on the report is the consolidated line for the parent value, followed by the consolidated lines for each of its children, to the level specified by *n*.

n|ALL

Is a positive integer from 1 to 99, specifying the number of levels of children to display. The line of output for each child has the sum of that child and its children to the depth specified for the ADD option. The default value is 1. Therefore, if *n* is omitted, each direct child has a line on the report. If *n* is 2, direct children and grandchildren each have a line on the report output. ALL is a synonym for 99.

ADD

Sums the hierarchy to the depth specified by *m* for each line generated by the GET or WITH CHILDREN command.

m|ALL

Is a positive integer from 1 to 99, specifying the number of levels of children to consolidate on each line of the report output. If a number greater than 99 is specified, a warning message is displayed and *m* is set to 99. The default value is ALL. Therefore, if *m* is omitted, the consolidated line sums all children. If *m* is 2, only direct children and grandchildren are consolidated for each line on the report output. ADD 99 aggregates children to 99 levels. ALL is a synonym for 99.

CAPTION

Indicates that the caption of the parent value displays for the total row.

Note that the AS CAPTION phrase is supported for tagged rows, including those that do not use the GET CHILDREN or ADD syntax. However, the hierarchy must be defined (by specifying the PARENT_OF attribute) in order to load and display the caption values. If the hierarchy is not defined, the AS CAPTION phrase is ignored.

'text'

Is a text string to use as the row title for the aggregate row. The CAPTION field defined in the Master File is not used as the caption on the report output.

label

Is an explicit row label. Each generated row is labeled with the specified label text.

Example: Consolidating FML Hierarchy Data

In the following request, the first WITH CHILD command displays the detail data for the hierarchy starting with account 3100. The next WITH CHILD command creates a consolidated line for the parent account (3100) and each direct child:

```

SET FORMULTIPLE = ON
JOIN SYS_ACCOUNT IN CENTGL TO ALL SYS_ACCOUNT IN CENTSYSF
TABLE FILE CENTGL
SUM NAT_AMOUNT/D10.0 NAT_YTDAMT/D10.0
FOR GL_ACCOUNT
3100 WITH CHILDREN ALL AS CAPTION          OVER
" "                                       OVER
BAR AS =                                OVER
" "                                       OVER
3100 WITH CHILDREN  ADD AS CAPTION
IF PERIOD EQ '2002/03'
END
    
```

Note that the join is not required to be unique, because the hierarchy is defined in the host segment.

In the following output, the top portion shows the detail-level data. The bottom portion shows the consolidated data. In the consolidated portion of the report:

- ❑ There is one line for the parent that is the sum of itself plus all of its children to all levels.
- ❑ There is one line for each direct child of account 3100 (Selling Expenses): Advertising, Promotional Expenses, Joint Marketing, and Bonuses/Commissions.
- ❑ The line for Advertising is the sum of itself plus all of its children. If it has multiple levels of children, they are all added into the sum. The other direct children of 3100 do not themselves have children, so the sum on each of those lines consists of only the parent value.

	Month Actual -----	YTD Actual -----
Selling Expenses	.	.
Advertising	.	.
TV/Radio	1,049,146.	2,954,342.
Print Media	244,589.	721,448.
Internet Advertising	9,542.	29,578.
Promotional Expenses	53,719.	151,732.
Joint Marketing	97,135.	289,799.
Bonuses/Commissions	100,188.	304,199.
	=====	=====
Selling Expenses	1,554,319.	4,451,098.
Advertising	1,303,277.	3,705,368.
Promotional Expenses	53,719.	151,732.
Joint Marketing	97,135.	289,799.
Bonuses/Commissions	100,188.	304,199.

Using GET CHILDREN instead of WITH CHILDREN eliminates the top line from each portion of the output. The remaining lines are the same:

	Month Actual -----	YTD Actual -----
Advertising	.	.
TV/Radio	1,049,146.	2,954,342.
Print Media	244,589.	721,448.
Internet Advertising	9,542.	29,578.
Promotional Expenses	53,719.	151,732.
Joint Marketing	97,135.	289,799.
Bonuses/Commissions	100,188.	304,199.
	=====	=====
Advertising	1,303,277.	3,705,368.
Promotional Expenses	53,719.	151,732.
Joint Marketing	97,135.	289,799.
Bonuses/Commissions	100,188.	304,199.

The following request displays a consolidated line for account 2000 and each of its direct children and grandchildren.

```

SET FORMULTIPLE = ON
JOIN SYS_ACCOUNT IN CENTGL TO ALL SYS_ACCOUNT IN CENTSYSF
TABLE FILE CENTGL
SUM NAT_AMOUNT/D10.0 NAT_YTDAMT/D10.0
FOR GL_ACCOUNT
2000 WITH CHILDREN 2 ADD AS CAPTION
IF PERIOD EQ '2002/03'
END
    
```

The output is:

	Month Actual -----	YTD Actual -----
Gross Margin	-4,513,659.	-13,080,549.
Sales Revenue	-10,398,305.	-30,877,546.
Retail Sales	-8,237,253.	-24,539,197.
Mail Order Sales	-1,138,414.	-3,403,387.
Internet Sales	-1,022,638.	-2,934,962.
Cost Of Goods Sold	5,884,646.	17,796,997.
Variable Material Costs	4,415,560.	13,410,629.
Direct Labor	961,143.	2,920,449.
Fixed Costs	507,943.	1,465,919.

Loading a Hierarchy Manually

How to:

Load a Hierarchy From One Master File for Use With a Separate Master File

Reference:

Usage Notes for FML Hierarchies

In most cases, a hierarchy is loaded automatically as a result of the request syntax. However, if you need to use a hierarchy defined in one Master File against a data source that is not joined to the hierarchy file (but that contains the same hierarchy field), you can manually load the hierarchy data using the LOAD CHART command.

The number of charts that can be loaded is limited by available memory. Charts are automatically unloaded when the session ends.

The chart is loaded by running a TABLE request that produces a list of parent values and their associated children:

```

TABLE FILE chartfile
BY parentfield BY hierarchyfield
[SUM captionfield]
END
    
```

The resulting chart contains the following information. It may also contain the associated captions, depending on whether the AS CAPTION phrase was used in the request:

```

parentfield      hierarchyfield
-----
parentvalue1    child1
parentvalue1    child2
parentvalue1    child3
.
.
.

```

Syntax: How to Load a Hierarchy From One Master File for Use With a Separate Master File

You can manually load the hierarchy data if you need to use a hierarchy defined in one Master File against a data source that is not joined to the hierarchy file but that contains the same hierarchy field.

The number of charts that can be loaded is limited by available memory. Charts are automatically unloaded when FOCUS terminates.

```

LOAD CHART chartfile[.sega].hierarchyfld
  [FOR requestfile[.segb].fieldb]]

```

where:

chartfile

Is the name of the Master File that contains the hierarchy information.

sega

Is the name of the segment that contains the hierarchy field. The segment name is only required if a field in another segment in the structure has the same field name as the hierarchy field.

hierarchyfld

Is the hierarchy field. It is required because a Master File can define multiple hierarchies.

FOR

Loads a hierarchy defined in a Master File that is not used in the FML report request. For example, if Master File B contains the hierarchy information but Master File A is used in the request (without a join between Master Files A and B), issue the following LOAD CHART command prior to the FML request:

```

LOAD CHART B.FLDB FOR A.FLDA
TABLE FILE A ...

```

requestfile

Is the name of the Master File used in the FML request.

segb

Is the name of the segment that contains the hierarchy field values in the Master File used in the FML request. Not required if it has the same name as *sega*.

fieldb

Is the field in the Master File specified in the FML request that contains the values of the hierarchy field. Not required if it has the same name as the hierarchy field.

Note:

- ❑ If you issue the LOAD CHART command multiple times for the same hierarchy, the new hierarchy overlays the previous version in memory.
- ❑ If you issue the LOAD CHART command for a data source that is dynamically joined to the hierarchy file, you must issue the JOIN command prior to issuing the LOAD CHART command.

Reference: Usage Notes for FML Hierarchies

- ❑ PROPERTY and REFERENCE are propagated to HOLD Master Files when HOLDATTR is set to ON.
- ❑ The following setting is required in order to use a data record in more than one row of an FML request (for example, both a detail and summary row):

```
SET FORMULTIPLE = ON
```
- ❑ When reporting against a rolled-up data source such as ESSBASE, the data values stored for the parent instance are an aggregate of all of its children. Do not use the ADD feature on consolidated data.
- ❑ When reporting against a data source with shared members (such as ESSBASE), in which the same data can be defined multiple times with different hierarchy field values, data shared by two different parents is counted twice in an aggregation operation. To avoid this double aggregation, use the FST. operator in the SUM command for the shared fields.

Customizing a Row Title

How to:

Customize a Row Title in FML

You can customize a row title in an FML report for accurate data identification. Using the AS phrase, you can provide new titles for TAG, DATA, RECAP, and PICKUP rows.

Syntax: How to Customize a Row Title in FML

For a TAG row, use the syntax:

```
value AS 'title'|CAPTION}
```

For a DATA or PICKUP row, use the syntax:

```
value AS 'title'
```

For a RECAP row, use the syntax:

```
RECAP calcname[/format]=expression; AS 'title'
```

where:

value

Is the value on which you are reporting, whether retrieved from a data source or external file (represented by a tag), supplied directly by a user in the request, or picked up from a work file.

calcname

Is the value that is derived by the RECAP calculation.

title

Is the customized row title, enclosed in single quotation marks if it contains embedded blanks.

In a TAG, DATA, or PICKUP row, the default row title is *value*.

In a RECAP row, the default title is *calcname*.

CAPTION

In the Master File of a hierarchical data source, CAPTION identifies a TAG row using a caption field. Note that the hierarchy in the Master File defines the PARENT-OF the FOR field.

Example: Changing the Titles of Tag Rows

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 since the row title being assigned has embedded blanks.

The output is:

	AMOUNT

CASH ON HAND	8,784
DEMAND DEPOSITS	4,494

If no AS phrase is included, the tag values are displayed in the report.

Example: Customizing a Row Title for a RECAP Value

This request creates the title TOTAL CASH for the RECAP value TOTCASH:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND'      OVER
1020 AS 'DEMAND DEPOSITS'  OVER
1030 AS 'TIME DEPOSITS'    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

If no AS phrase are included, the name of the RECAP value (TOTCASH) is displayed in the report.

Formatting an FML Report

In this section:

Indenting Row Titles in an FML Hierarchy

How to:

Add an Underline Character for Columns

Specify a Page Break in an FML Report

Specify an Indent for an FML Label, Tag, or Caption

Improve the readability and presentation of your FML report by:

- ❑ **Underlining numeric columns.** 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.

- ❑ **Adding 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.
- ❑ **Indenting text or numbers.** You can indent a tag value, label text, or caption text a specified number of spaces for an FML tag row, hierarchy, or RECAP row. If you apply the indent to rows in an FML hierarchy, the parent line of the hierarchy is indented the number of spaces specified as the indent.

Note: For an HTML, PDF, or postscript report, you can use the BLANKINDENT setting to specify an indentation between levels of an FML hierarchy. See *Indenting Row Titles in an FML Hierarchy* on page 997.

Note: You can also format an FML report using StyleSheet attributes if you are creating an output format that supports 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: Underlining 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 an FML Report

Include the following syntax in the FML request in place of a tag value:

PAGE-BREAK OVER

Example: Specifying a Page Break in an FML Report

In this example, a page break is inserted after the first two RECAP commands to highlight each calculation.

```
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'         OVER
PAGE-BREAK                                          OVER
1100 AS 'ACCOUNTS RECEIVABLE' LABEL RECEIVE         OVER
1200 AS 'INVENTORY' LABEL INVENT                    OVER
BAR                                                    OVER
RECAP TOTASSET = RECEIVE + INVENT; AS 'TOTAL ASSETS' OVER
PAGE-BREAK                                          OVER
RECAP TOTAL = TOTCASH + TOTASSET;
END
```

The output is:

```
PAGE      1
          AMOUNT
          -----
CASH ON HAND      8,784
DEMAND DEPOSITS   4,494
TIME DEPOSITS     7,961
          -----
TOTAL CASH        21,239
PAGE      2
          AMOUNT
          -----
ACCOUNTS RECEIVABLE 18,829
INVENTORY           27,307
          -----
TOTAL ASSETS        46,136
PAGE      3
          AMOUNT
          -----
TOTAL               67,375
```

Syntax: How to Specify an Indent for an FML Label, Tag, or Caption

`FOR forfield [IN k]`

`tag [[GET CHILDREN|WITH CHILDREN] n] INDENT m [AS ['text'|CAPTION]]
[OVER]`

or

`RECAP fieldname[/format]=expression; INDENT m [AS 'text']`

where:

k

Is the starting column for the FOR value in an FML report.

forfield

Is a field in the data source whose values are included in the report.

tag

Is a value of *forfield* to be displayed on a row of the FML report.

n

Is the number of levels of an FML hierarchy to display on the FML report.

m

Is a positive integer (zero is not supported) specifying the number of spaces to indent the tag value, label, or caption of an FML row or hierarchy. The indentation starts from column one if there is no IN phrase specified in the FOR command. If there is an IN phrase, indentation starts from the column specified in the IN phrase. The maximum indentation is the same as the maximum length of an AS name.

If you indent an FML hierarchy, the parent line of the hierarchy is indented the number of spaces specified as the indent. The hierarchy levels are indented two spaces from each other. If the GET CHILDREN phrase is used, the first line of the hierarchy is indented an additional two spaces because the hierarchy output begins with the first child rather than the parent. For more information about the use of GET CHILDREN, see [Displaying an FML Hierarchy](#) on page 978.

text

Is a label to be displayed on a row of the FML report.

`CAPTION`

Indicates that a caption field has been defined in the Master File.

`OVER`

Indicates that this row is not the last row to be displayed.

fieldname

Is a name you assign to the value calculated by the RECAP command.

format

Is the USAGE format for RECAP field. It cannot exceed the column width. The default is the format of the column in which the calculated value is displayed.

expression

Is the expression that describes how to calculate the field value for RECAP.

Example: Indenting a Tag Row in an FML Hierarchy

In the following request, the label of the second row for tag value 3000 is indented five spaces. Because the GET CHILDREN phrase is used, the first line of the FML hierarchy, in the third row for tag value 3000, is indented seven spaces (five + two):

```
SET FORMULTIPLE = ON
TABLE FILE CENTGL
PRINT GL_ACCOUNT_PARENT
FOR GL_ACCOUNT
3000 AS 'Not Indented' OVER
3000 INDENT 5 AS 'Indented 5' OVER
3000 GET CHILDREN 2 INDENT 5 AS 'Hierarchy Indented 5'
END
```

The output is:

	Parent

Not Indented	3000
Indented 5	3000
Hierarchy Indented 5	3000
Hierarchy Indented 5	3100
Hierarchy Indented 5	3000
Hierarchy Indented 5	3200

Example: Indenting FML RECAP Rows

The following request sums price, cost, and quantity in stock for digital and analog product types. The first RECAP command calculates the total for each column, and indents the label five spaces. The second RECAP command calculates the profit, and indents the label 10 spaces:

```
SET FORMULTIPLE = ON
TABLE FILE CENTINV
SUM PRICE COST QTY_IN_STOCK
FOR PRODTYPE
Digital                                OVER
Analog                                OVER
BAR                                    OVER
RECAP TOTAL = R1 + R2; INDENT 5 AS 'Total:' OVER
BAR                                    OVER
RECAP PROFIT(2) = TOTAL(1) - TOTAL(2); AS 'Profit:' INDENT 10
END
```

The output is:

	Price:	Our Cost:	Quantity In Stock:
	-----	-----	-----
Digital	4,080.00	3,052.00	119143
Analog	1,883.00	1,371.00	139345
	-----	-----	-----
Total:	5,963.00	4,423.00	258488
	-----	-----	-----
Profit:		1,540.00	

Indenting Row Titles in an FML Hierarchy**How to:**

Indent FML Hierarchy Captions in an HTML Report

To clarify relationships within an FML hierarchy, the captions (titles) of values are indented at each level. Use the BLANKINDENT parameter in an HTML, PDF, or PostScript report to specify the indentation between each level in the hierarchy. You can use the default indentation for each hierarchy level, or choose your own indentation value. To print indented captions in an HTML report, you must set the BLANKINDENT parameter to ON or to a number.

SET BLANKINDENT does not increase the width of the indented column if it is not wide enough to accommodate the indented fields. While this is no problem in an HTML report, in a PDF or PostScript report it can cause data in the columns that follow the indented column to shift out of alignment. You may need to use StyleSheet syntax to make the column wide enough for the indented values or to move the columns that follow it. Change the width of a column using the StyleSheet SQUEEZE attribute, and specify a starting position for a column using the POSITION attribute. You can also move a column in a PostScript report with the IN phrase.

A related feature enables you to change the number of blank spaces before the parent line of a hierarchy or before any FML tag or RECAP row in any FML request. See [Formatting an FML Report](#) on page 992.

Syntax: **How to Indent FML Hierarchy Captions in an HTML Report**

```
SET BLANKINDENT = {ON|OFF|n}  
ON TABLE SET BLANKINDENT {ON|OFF|n}
```

where:

ON

Indents FML hierarchy captions 0.125 units for each space that normally displays before the caption. For child levels in an FML hierarchy, it indents 0.125 units for each space that normally displays between this line and the line above it.

OFF

Turns off indentations for FML hierarchy captions in an HTML report. OFF is the default value. For other formats, uses the default indentation of two spaces.

n

Is an explicit measurement in the unit of measurement defined by the UNITS parameter. This measurement is multiplied by the number of spaces that normally displays before the caption. For child levels in an FML hierarchy, it indents *n* units for each space that normally displays between this line and the line above it. The default number of spaces is two. Zero (0) produces the same report output as OFF. Negative values for *n* are not supported.

Example: Using the Default Indentation for FML Hierarchy Captions

The following request creates an HTML report with the default indentation:

```

SET PAGE-NUM = NOPAGE
SET BLANKINDENT = ON
SET FORMULTIPLE = ON
TABLE FILE CENTGL
PRINT GL_ACCOUNT_PARENT
FOR GL_ACCOUNT
3000                               AS CAPTION      OVER
3000 GET CHILDREN 2 AS CAPTION      ON
TABLE HOLD FORMAT HTML
ON TABLE SET HTMLCSS ON
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
ENDSTYLE
END

```

The output is:

	<u>Parent</u>
Total Operating Expenses	1000
Selling Expenses	3000
Advertising	3100
Promotional Expenses	3100
Joint Marketing	3100
Bonuses/Commissions	3100
General + Admin Expenses	3000
Salaries-Corporate	3200
Company Benefits	3200
Depreciation Expenses	3200
Gain/(Loss) Sale of Equipment	3200
Leasehold Expenses	3200
Interest Expenses	3200
Utilities	3200

Example: Specifying an Indentation Value for FML Hierarchy Captions

The following request specifies an indentation of .25 for each level of an FML hierarchy. This number is expressed in the default unit of measurement, inches:

```
SET PAGE-NUM = NOPAGE
SET BLANKINDENT = .25
SET FORMULTIPLE = ON
TABLE FILE CENTGL
PRINT GL_ACCOUNT_PARENT
FOR GL_ACCOUNT
3000 AS CAPTION OVER
3000 GET CHILDREN 2 AS CAPTION ON
TABLE HOLD FORMAT HTMLON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, $
ENDSTYLE
END
```

The output is:

	<u>Parent</u>
Total Operating Expenses	1000
Selling Expenses	3000
Advertising	3100
Promotional Expenses	3100
Joint Marketing	3100
Bonuses/Commissions	3100
General + Admin Expenses	3000
Salaries-Corporate	3200
Company Benefits	3200
Depreciation Expenses	3200
Gain/(Loss) Sale of Equipment	3200
Leasehold Expenses	3200
Interest Expenses	3200
Utilities	3200

Suppressing the Display of Rows

In this section:

Suppressing Rows With No Data

You may sometimes wish to retrieve data in a TAG row solely for use in a calculation, without displaying the row in a report. To suppress the display of a tag row, add the word NOPRINT to the row declaration, as in a TABLE request.

You may also wish to suppress the display of a TAG row if no data is found for the values. See [Suppressing Rows With No Data](#) on page 1002.

In addition, you can suppress the display of RECAP rows by adding the word NOPRINT to the RECAP command, following the semicolon. This technique is useful to suppress the display of an intermediate RECAP value, which is intended for use as input to other calculations.

Example: Suppressing the Display of a TAG Row

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 tag 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 TAG row. This makes displaying a row dependent upon the existence of data for the tag. This feature is useful, for example, when the same model is applied to different divisions in a company.

Example: Suppressing Rows With No Data

The CENTSYSF data source has detail-level financial data. Accounts with no children are populated, but those with children are not. The following request suppresses the display of accounts that are not populated:

```
SET FORMULTIPLE = ON
JOIN SYS_ACCOUNT IN CENTGL TO ALL SYS_ACCOUNT IN CENTSYSF
TABLE FILE CENTGL
SUM NAT_AMOUNT/D10.0 NAT_YTDAMT/D10.0
FOR GL_ACCOUNT
3100 WITH CHILDREN ALL AS CAPTION WHEN EXISTS
IF PERIOD EQ '2002/03'
END
```

The output is:

	Month Actual	YTD Actual
	-----	-----
TV/Radio	1,049,146.	2,954,342.
Print Media	244,589.	721,448.
Internet Advertising	9,542.	29,578.
Promotional Expenses	53,719.	151,732.
Joint Marketing	97,135.	289,799.
Bonuses/Commissions	100,188.	304,199.

Saving and Retrieving Intermediate Report Results

In this section:

Posting Data

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 Windows, UNIX, and CMS, or a TSO ALLOCATE or DYNAM ALLOCATE command on z/OS, before the request is run. You may create a FILEDEF command by using the Allocation Wizard.

While you cannot prepare an FML report entirely from data that you supply directly in your request, you can prepare a report entirely from data that is stored in a comma-delimited work file.

Posting Data

How to:

Post Data to a File

Pick Up Data From a Work File

You can save any TAG, 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 is 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 (\$). See [Posting Rows to a Work File](#) on page 1004.

Note: Only fields that are actually displayed on the report output are posted. Fields that are not printed (for example, fields specified with the NOPRINT option, extra fields that are created when you re-format fields in the request, or fields implied by use in a calculation) are not posted.

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.

Add this syntax to any row you wish to post to the work file.

Example: Posting Rows to a Work File

The following request creates an FML report, and posts two tag rows to the work file, LEDGEOUT:

```
CMS FILEDEF LEDGEOUT DISK LEDGEOUT DATA A

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:

	CUR_YR	LAST_YR
	-----	-----
1100	18,829	15,954
1200	27,307	23,329

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.

Note: The retrieved fields are mapped to all fields (printed or not) in the memory repository (internal matrix) of the report. If the matrix contains columns that do not correspond to the fields in the posted file, the retrieved values may be misaligned. For example, if you reformat a field in the PICKUP request, that field will be represented by two columns in the internal matrix. However, the posted file will have only one value representing that field, and the retrieved values will not be mapped properly to the associated columns in the matrix.

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 are used in the RECAP calculation. (To see how this file was created, refer to [Posting Rows to a Work File](#) on page 1004.)

Tip: You must assign a logical name to the file by issuing a FILEDEF command on Windows, UNIX, and CMS, or a DYNAM ALLOCATE command on z/OS, before the request is run. You may create a FILEDEF command by using the Allocation Wizard.

```
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 can 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 can be 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.

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
```

Query the HOLD file:

```
>
? hold

DEFINITION OF HOLD FILE: HOLD

FIELDNAME          ALIAS          FORMAT
                   E01          A 19
AMOUNT             E02          I5C
```

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

20 | Creating 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.

Note: You can create free-form reports with PDF, HTML, and Styled formats. HTML output has all report pages on one HTML page. Page breaks are retained in PDF output.

Topics:

- ❑ Creating a Free-Form Report
- ❑ Designing a Free-Form Report

Creating a Free-Form Report

You can create a free-form report from 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:

Heading	Contains the body of the report. It displays the text characters, graphic characters, and data fields that make up the report.
Footing	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.
Prefix operators	Indicates field calculations and manipulation.
Temporary fields	Derives new values from existing fields in a data source.
BY phrases	Specifies the report sort order, and determines how many records are included on each page.
WHERE criteria	Selects records for the report.

When creating a free-form report, you can:

- ❑ Design your report to include text, data fields, and graphic characters. See [Designing a Free-Form Report](#) on page 1014.
- ❑ Customize the layout of your report. See [Laying Out a Free-Form Report](#) on page 1016.
- ❑ Select the sort order and the records that are included in your report. See [Sorting and Selecting Records in a Free-Form Report](#) on page 1016.

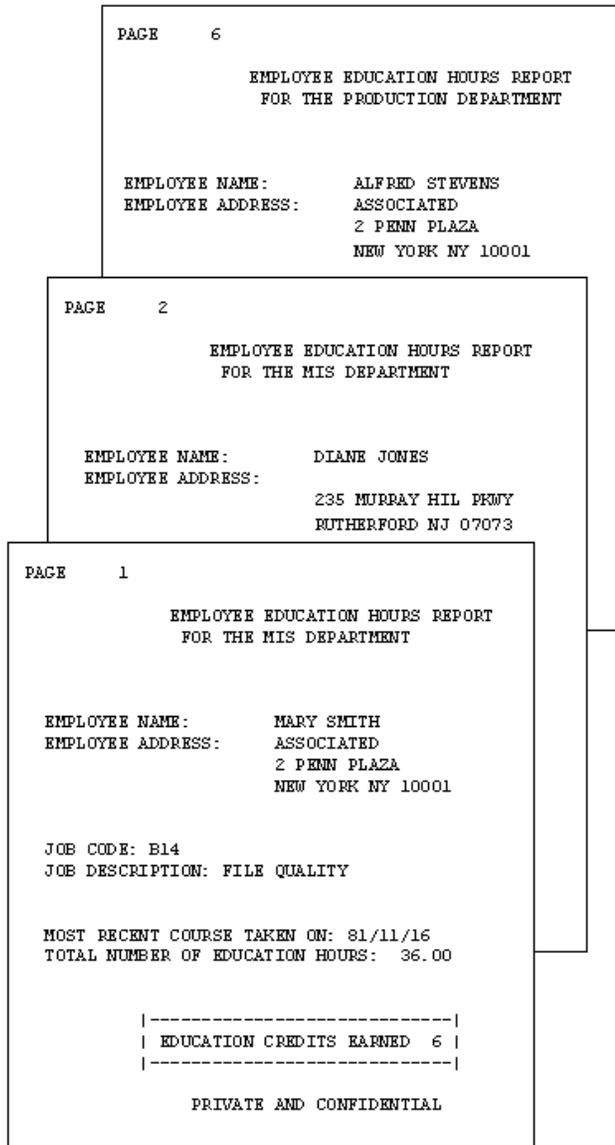
Example: Creating a Free-Form Report

Suppose that you are a Personnel Manager and it is your responsibility to administer your company education policies. This education policy states that the number of hours of outside education that an employee may take at the company expense is determined by the number of hours of in-house education completed by the employee.

To do your job efficiently, you want a report that shows the in-house education history of each employee. Each employee information should display on a separate page so that it can be placed in the employee 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 1012.



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* on page 1010. 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 NOPRINT
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 the versatility of 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.** Line-skipping commands enhance the readability of a report. 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. The request here is for the date on which the most recent course was taken, which 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. This is the field that was created at the start of the request.
- 9.** The `FOOTING` command signifies the beginning of the footing section, ending the heading section as well. Since this is a personnel report, the words `PRIVATE AND CONFIDENTIAL` must appear at the end of each page of the report. The footing can accomplish this.
- 10.** This line illustrates 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

In this section:

- Incorporating Text in a Free-Form Report
- Incorporating Data Fields in a Free-Form Report
- Incorporating Graphic Characters in a Free-Form Report
- Laying Out a Free-Form Report
- Sorting and Selecting Records in a Free-Form Report

To design the body of a free-form report, use the **HEADING** and **FOOTING** commands. They enable you to:

- ❑ 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 is optional. You can 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 (see [Request for EMPLOYEE EDUCATION HOURS REPORT](#) on page 1012) 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 in a request include data fields in the Master File, cross-referenced fields, and virtual fields created with the DEFINE command.

The sample request (see [Request for EMPLOYEE EDUCATION HOURS REPORT](#) on page 1012) 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

Graphics in a report can be as creative as your imagination. The sample report (see [Creating a Free-Form Report](#) on page 1010) uses special characters to enclose text and a virtual field in a box. Some other ideas include:

- ❑ Highlighting key data fields using asterisks or other special characters available directly from your keyboard, or using the HEXBYT function. 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 page heading and page 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 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, use the spot marker feature of the HEADING and FOOTING commands.

Note: To take advantage of this feature in an HTML report, include the SET STYLEMODE=FIXED command in your request.

The sample request (see [Request for EMPLOYEE EDUCATION HOURS REPORT](#) on page 1012) 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 (PDF output only), and UNDER-LINE. The sample request uses PAGE-BREAK to place each employee 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. Use the same commands as 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.

21

Creating Graphs: GRAPH

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 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.

Topics:

- ❑ Introduction
- ❑ Command Syntax
- ❑ Graph Forms
- ❑ Adjusting Graph Elements
- ❑ Special Topics
- ❑ Special Graphics Devices
- ❑ Command and SET Parameter Summary

Introduction

In this section:

- GRAPH vs. TABLE Requests
- Running Graph Requests Offline
- Controlling the Format
- Graphic Devices Supported

The following topics explain how to generate each graph form 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 of the examples 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 is 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:

```
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END
```

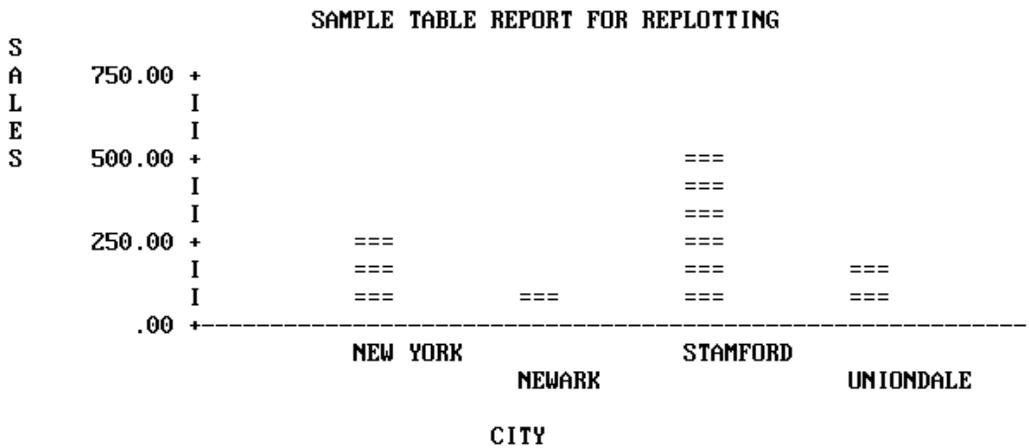
```
TABLE FILE SALES
HEADING CENTER
"SAMPLE TABLE REPORT FOR REplotTING"
SUM SALES ACROSS CITY
END
```


To produce the graph without creating a preliminary tabular report, substitute the command GRAPH for TABLE in the original request, as shown in the following:

```

DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END

GRAPH FILE SALES
HEADING CENTER
"SAMPLE TABLE REPORT FOR REPLOTTING"
SUM SALES ACROSS CITY
END
    
```

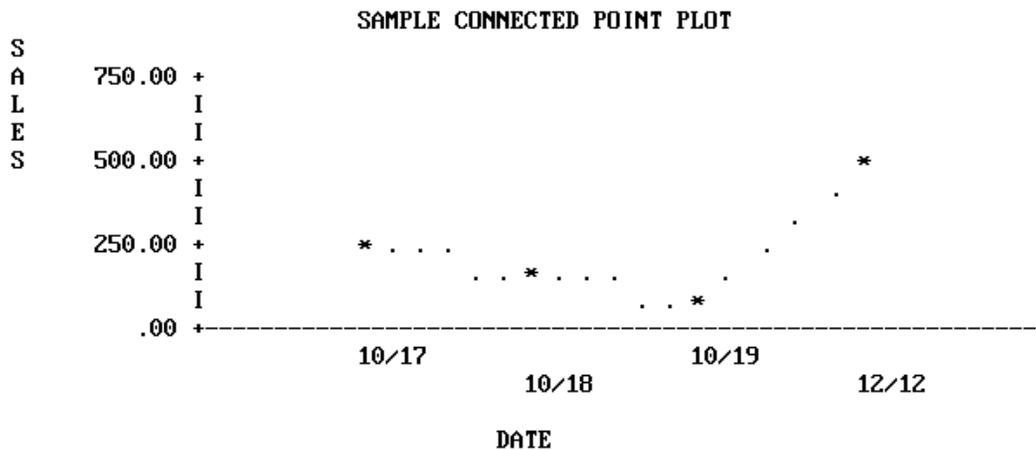


Thus, you can produce graphs by converting TABLE requests, but every TABLE facility does not have a GRAPH counterpart, and there are some practical limitations on the amount of information that you can effectively display in a graph. [Command Syntax](#) on page 1031 describes the use of TABLE features 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
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END
```

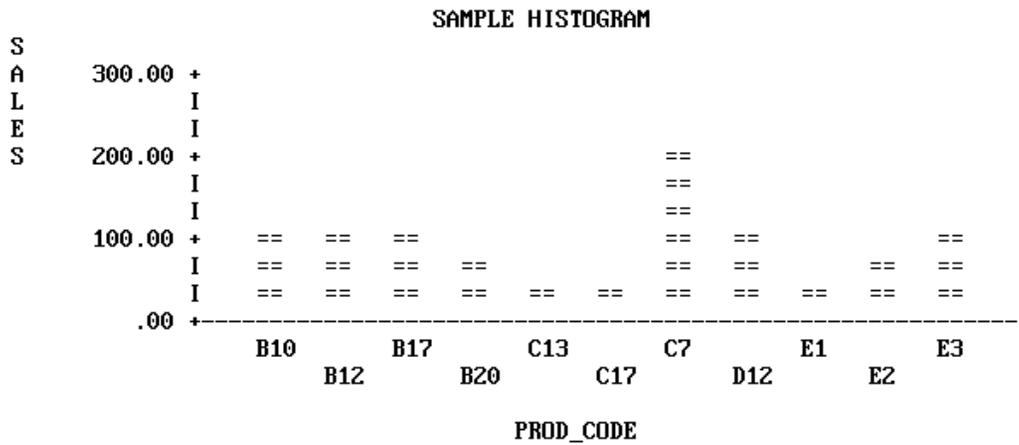
```
GRAPH FILE SALES
HEADING CENTER
"SAMPLE CONNECTED POINT PLOT"
SUM SALES ACROSS DATE
END
```



Note: SET parameters remain in effect until you reset them or log off (see [SET Parameters](#) on page 1081).

```
SET HISTOGRAM=ON
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END
```

```
GRAPH FILE SALES
HEADING CENTER
"SAMPLE HISTOGRAM"
SUM SALES ACROSS PROD_CODE
END
```



```

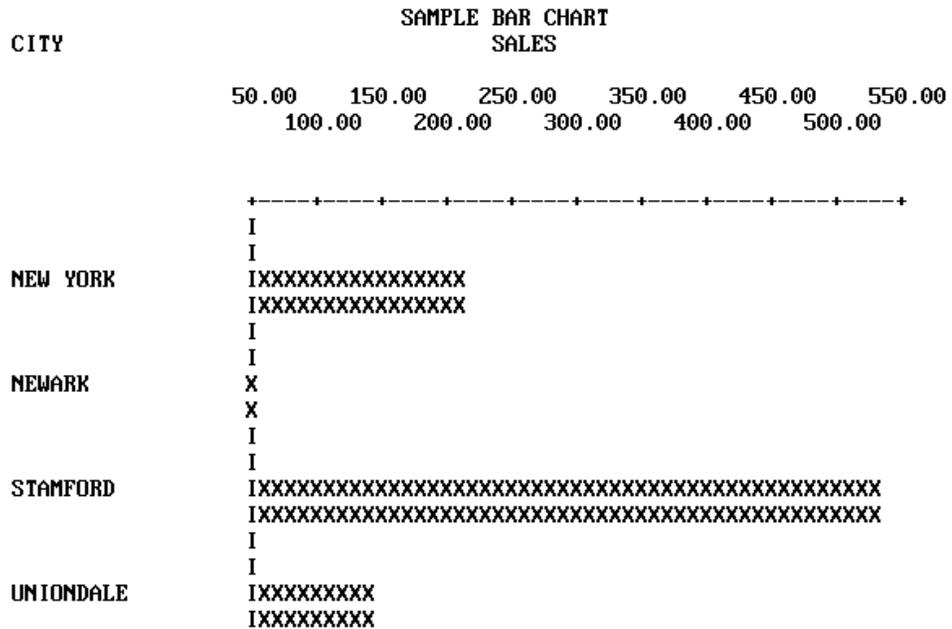
SET BARWIDTH=2, BARSPEACE=2
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END

```

```

GRAPH FILE SALES
HEADING CENTER
"SAMPLE BAR CHART"
SUM SALES BY CITY
END

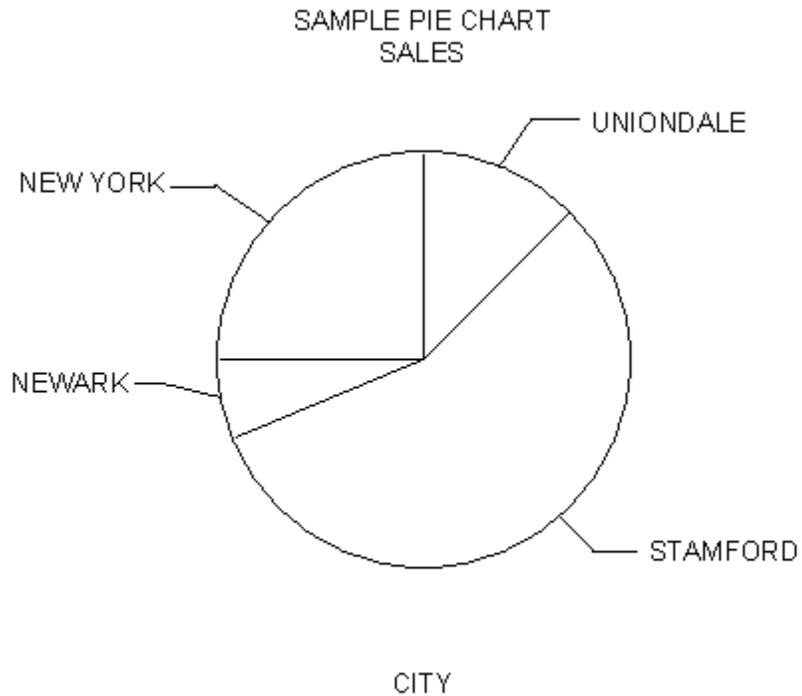
```



```
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
```

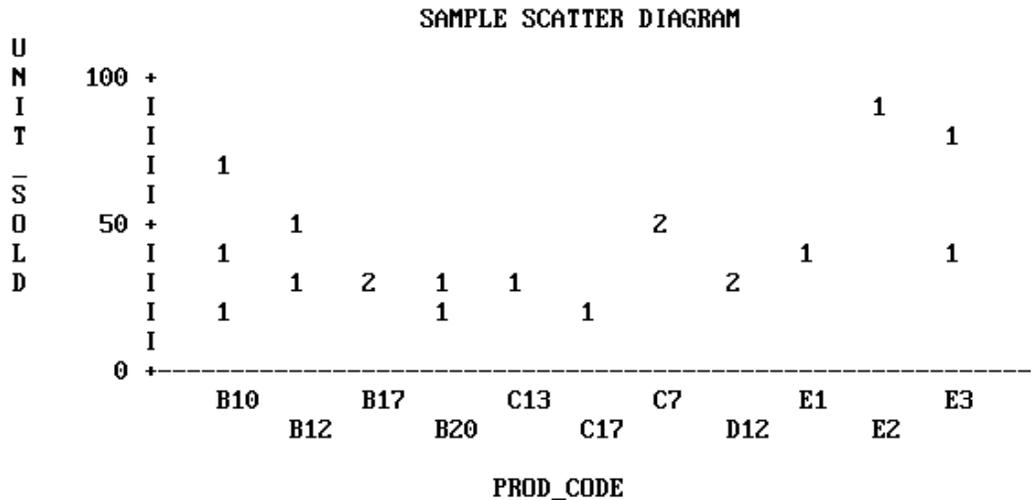


```

SET HISTOGRAM=OFF

GRAPH FILE SALES
HEADING CENTER
"SAMPLE SCATTER DIAGRAM"
PRINT UNIT_SOLD ACROSS PROD_CODE
END

```



Running Graph Requests Offline

How to:

Run a Graph Request Offline

Certain options in a graph request may prevent the graph from displaying properly on the screen. In this case, you can run the graph request offline, and spool the output to a sequential file.

Procedure: How to Run a Graph Request Offline

1. On z/OS create a sequential file to contain the graph output and allocate it to DDNAME OFFLINE. On CMS, FILEDEF a sequential file to DDNAME OFFLINE. For example:

```
DYNAM ALLOC DD OFFLINE DA USER1.OFFLINE.DATA SHR REU
```

or

```
CMS FILEDEF OFFLINE DISK OFFLINE DATA A
```

Note that the LRECL for the sequential file should be 132.

2. Issue the following command to route graph output to this file:

```
OFFLINE
```

If you want to run a request online after issuing the OFFLINE command, issue the ONLINE command:

```
ONLINE
```

You can then issue the OFFLINE command to run a request offline.

Each new graph request that is run offline appends its output to this file until you issue the following command:

```
OFFLINE CLOSE
```

Controlling the Format

In each of the previous graphs, FOCUS created a clear representation of the data using default values for the graph features (such as axis lengths, axis scales, or titles). You can issue your initial request and concentrate on selecting the data, while FOCUS controls all of the features on the graph.

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.

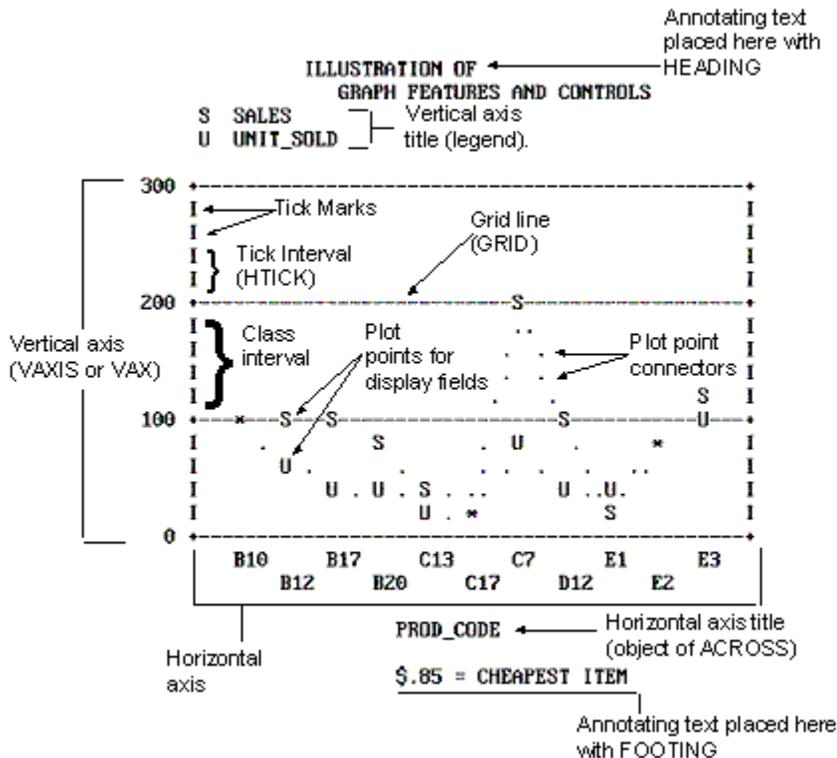
The request below illustrates some of the parameters you can control when running the graph offline:

```

SET HISTOGRAM=OFF
SET HAXIS=75, VAXIS=32, GRID=ON
SET AUTOTICK=OFF, VCLASS=100, VTICK=20
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END

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.



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 the previous figure, Illustration of Graph Features and Controls, in one of two ways: either by the syntax in the actual request, or with SET commands. [Command Syntax](#) on page 1031 describes the elements in GRAPH requests and their effects. [Adjusting Graph Elements](#) on page 1058 describes the adjustable parameters that control graph features.

There are 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 1081.

Graphic Devices Supported

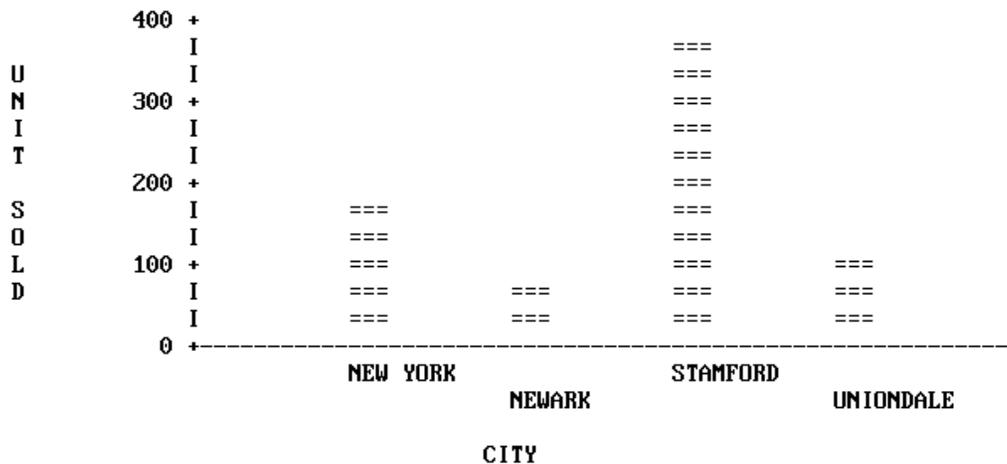
In this section:

Medium-Resolution Devices

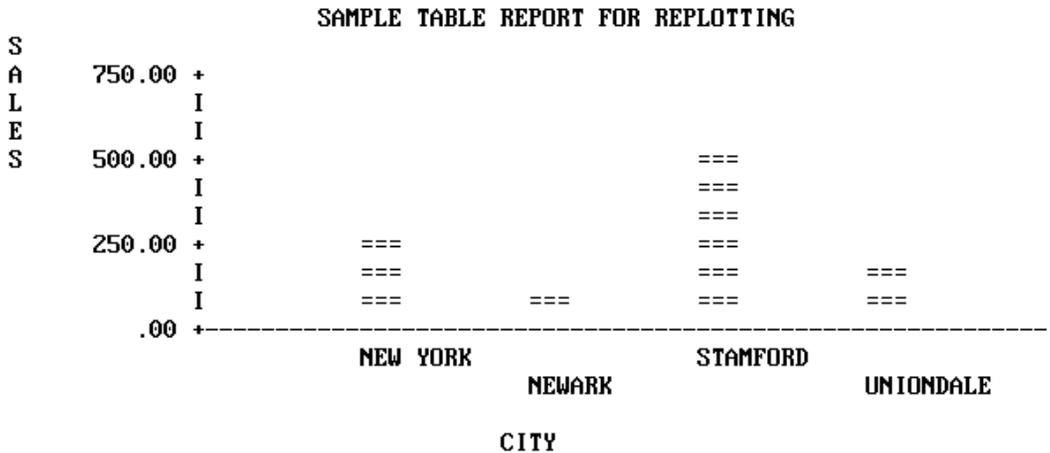
High-Resolution Devices

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. The following figure shows a graph on an IBM PC mono screen:



If color monitors or multiple-pen plotters are available, graph quality can be improved. The following figure shows a sample graph on a plotter. For more information, see [Special Graphics Devices](#) on page 1075.



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 1076. 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 1076). 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

In this section:

GRAPH vs. TABLE Syntax

Specifying Graph Forms and Contents

Most TABLE requests can be converted into GRAPH requests by replacing the TABLE command with the GRAPH command. The only limitations are those inherent in the graphic format. When a TABLE request is converted in this manner, the phrases that make up 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. It 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 1036.
- ❑ 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 1034).
- ❑ 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 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.
- ❑ 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 the functions performed by each of the phrases used in GRAPH requests.

Specifying Graph Forms and Contents

In this section:

Naming Subjects: Verb Phrases

Selecting Forms: BY and ACROSS Phrases

Selecting the Contents: Selection Phrases

Concatenating Unlike Data Sources

Adding Annotating Text: HEADING and FOOTING Lines

Inserting Formatting Controls

Inserting Field References

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 1018, are summarized in the table below (A and B represent two field names).

```
Point plot:  SUM A ACROSS B (B is numeric)
Histogram:  SET HISTOGRAM=ON
            SUM A ACROSS B (B is alpha)
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
```

Naming Subjects: Verb Phrases

Each GRAPH request must include a verb and at least one 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 1045), and Bar Charts (see [Bar Charts](#) on page 1048).

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 1063 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 1074. 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 1037).

Selecting the Contents: Selection Phrases

Selection phrases are used in GRAPH requests to select records of interest. Two 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

Relation	Meaning
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 [Merging Data Sources](#) on page 877, 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. 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
```

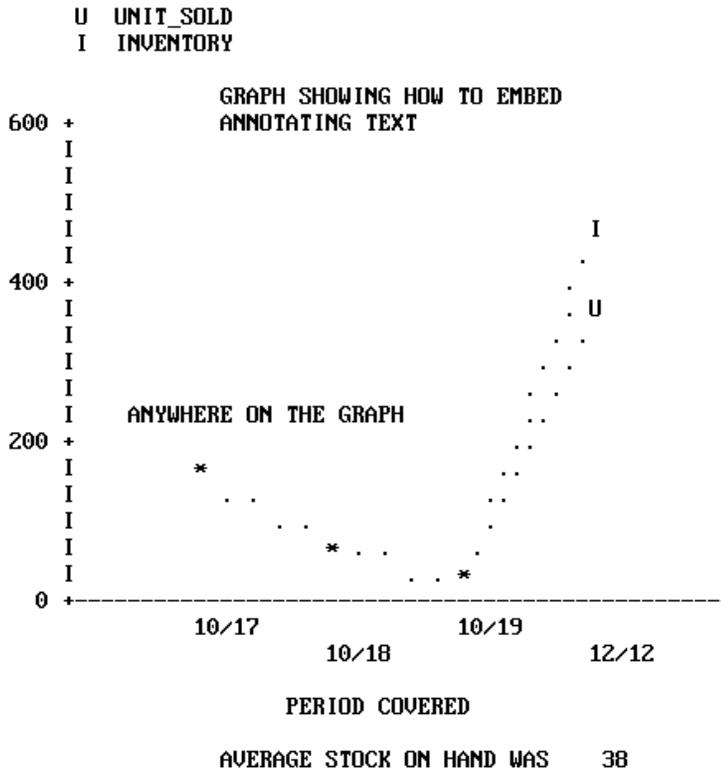
Note: When annotating text falls in the path of a plot point on a graph, the plot point is printed; however, connecting points are suppressed if they lie in the path of annotating text. This enables you to adjust the position of the annotating text when you see the contents of the graph. The first line 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 in the body of your graph. The following example shows the use of spot markers, which are described in Chapter 9, *Customizing Tabular Reports*. Run the following request offline to generate the graph shown immediately following the request:

```

SET HISTOGRAM=OFF
SET AUTOTICK=OFF, VCLASS = 200, VTICK = 25
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
    
```



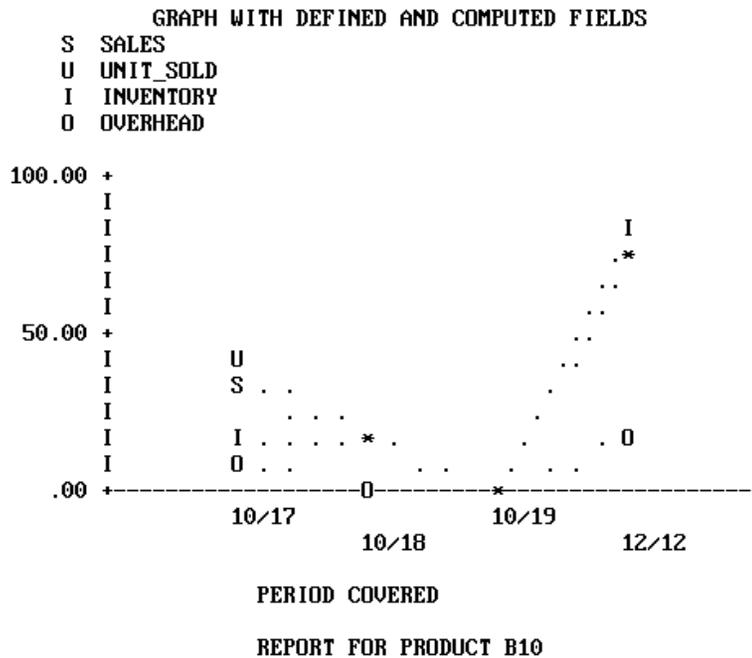
Inserting Field References

The following example shows how to embed field values in graph heading or footing lines, similar to the capability in TABLE requests. It is useful when annotating graphs created by requests containing multiple sort fields (where only the first named sort field appears as a title on the graph). Run the following as an offline request:

```

SET HISTOGRAM=OFF
SET AUTOTICK=OFF, VCLASS = 50 , VTICK = 8
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

In this section:

Connected Point Plots

Histograms

Bar Charts

Pie Charts

Scatter Diagrams

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 1058.

As seen in the examples in [GRAPH vs. TABLE Requests](#) on page 1018, there are similarities between the requests for some of the forms. For example, a request for a connected point plot (with an alphanumeric ACROSS field) creates a histogram instead if the HISTOGRAM parameter is set on (the default). This feature enables you retrieve data once, then switch 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.

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

In this section:

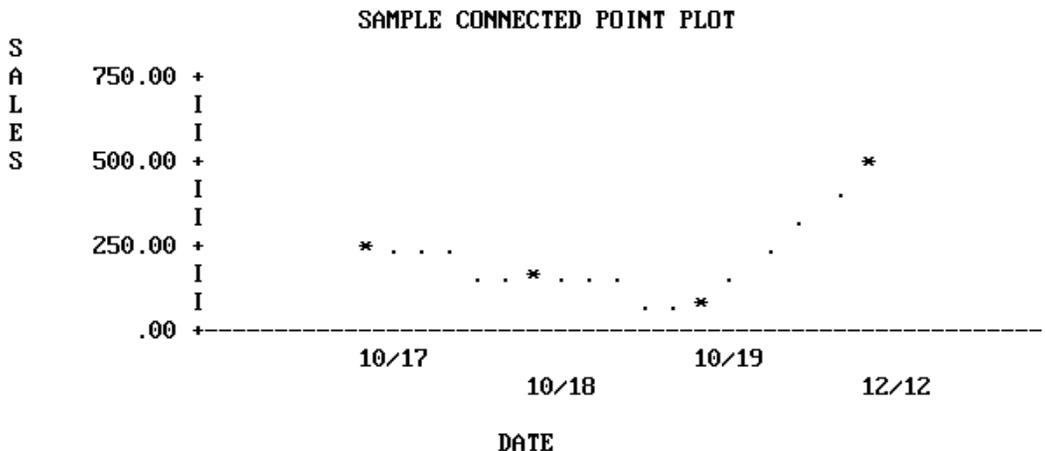
Point Plot Features

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 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, the normal page length.

Point Plot Features

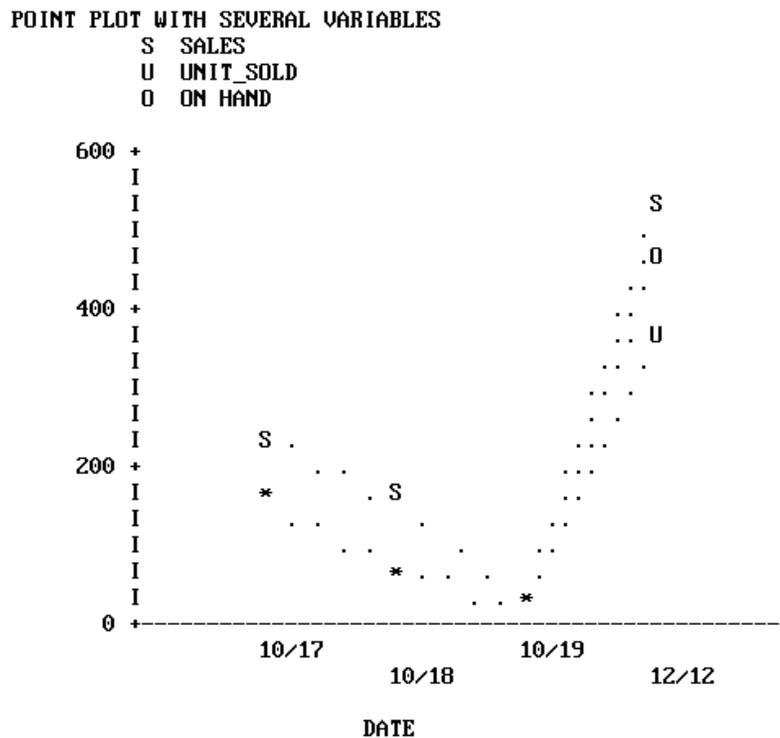
- ❑ **Scale Titles.** The values associated with the class markers are printed below the horizontal axis in the USAGE format of the variable being plotted (MM/DD in our example).
- ❑ **Plot Characters.** The graphics characters used to plot the variables on connected point plots depend on the type of display device:
 - ❑ 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 (rename duplicates with AS phrases). The data points are connected by periods (.). You cannot create continuous line plots, as they are only available on high-resolution devices.
 - ❑ On high-resolution displays, printers, and plotters, the lines connecting plot points are drawn explicitly. 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 following example illustrates a point plot with several variables, run offline.

```

SET HISTOGRAM=OFF
SET AUTOTICK=OFF, VCLASS = 200, VTICK = 25
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END
GRAPH FILE SALES
HEADING
"POINT PLOT WITH SEVERAL VARIABLES"
SUM SALES AND UNIT_SOLD AND INV AS 'ON HAND'
ACROSS DATE
END

```

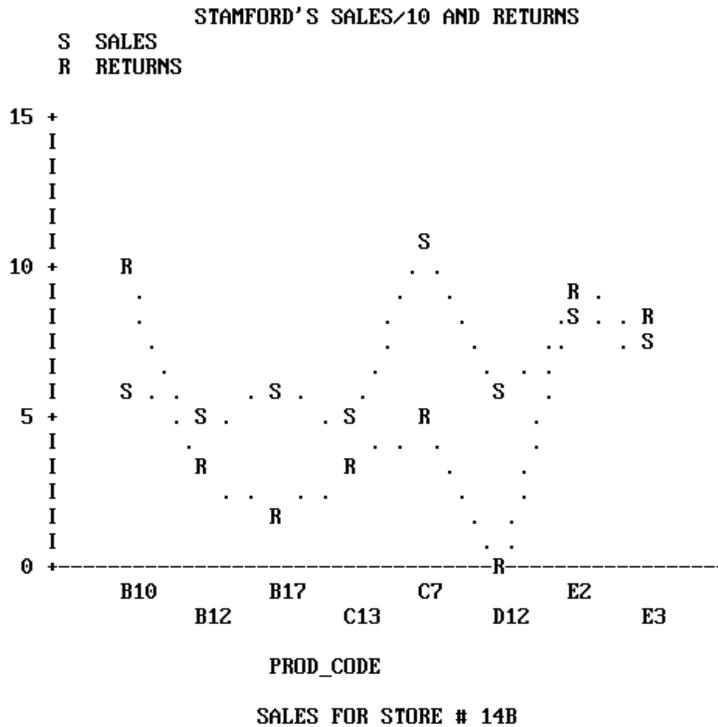


Up to five variables can be plotted on the same vertical axis. The scale on the vertical axis is determined based on the combined values of the vertical variables, and a separate point appears for each value of each variable.

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 make the finished graph more legible. A method for doing this is shown in the example below. Run this as an offline request:

```

DEFINE FILE SALES
SALES/D8.2=(UNIT_SOLD * RETAIL_PRICE)/10;
END
SET HISTOGRAM=OFF
SET AUTOTICK=OFF, VCLASS = 5 , VTICK = 1
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

In this section:

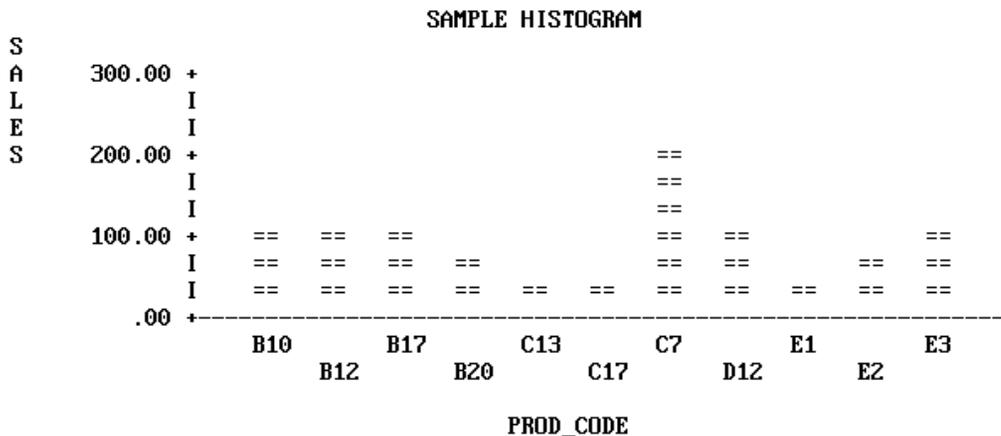
Histogram Features

Histograms are vertical bar charts, and are useful for portraying the component parts of aggregate values. They are an alternate graphic format for plotting requests that could also generate connected point plots. To switch from one format to the other, simply reset the parameter HIST and issue REPLOT.

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. Run it as an offline request:

```
SET HISTOGRAM=ON
SET AUTOTICK=OFF, VCLASS = 100, VTICK = 40
DEFINE FILE SALES
SALES/D8.2=(UNIT_SOLD * RETAIL_PRICE);
END

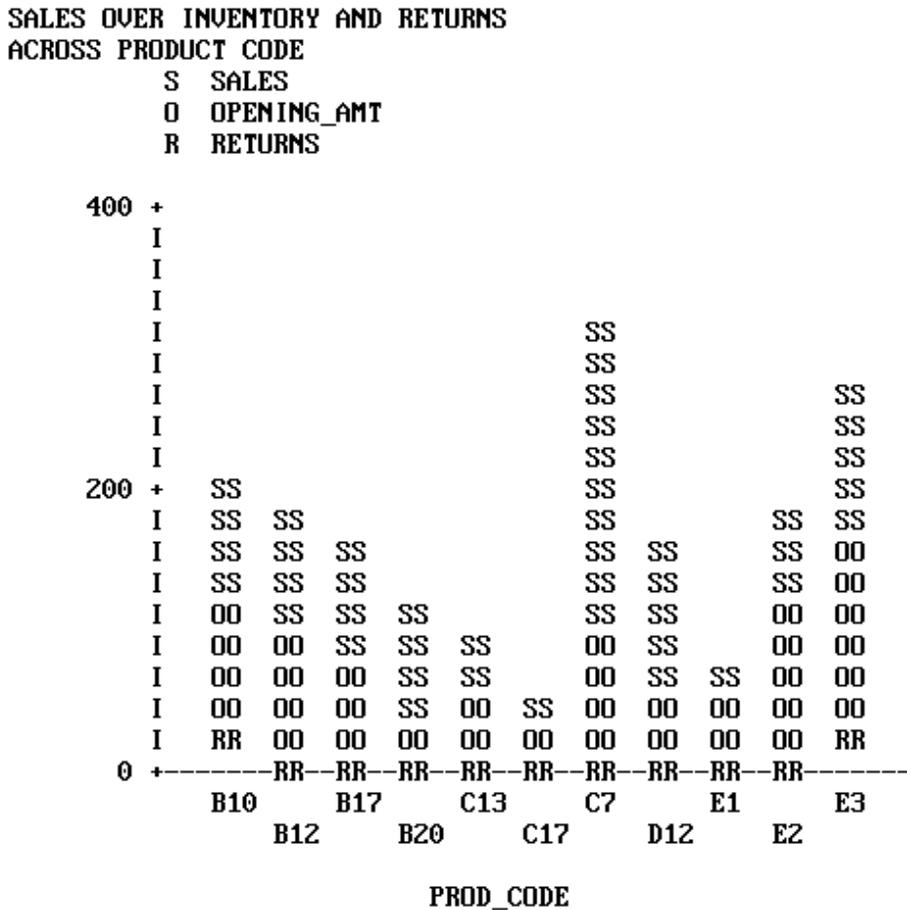
GRAPH FILE SALES
HEADING CENTER
"SAMPLE HISTOGRAM"
SUM SALES ACROSS PROD_CODE
END
```



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. Run it as an offline request:

```

SET HISTOGRAM = ON
SET AUTOTICK=OFF, VCLASS = 200, VTICK = 20
DEFINE FILE SALES
SALES/D8.2=(UNIT_SOLD * RETAIL_PRICE)    ;
END
GRAPH FILE SALES
HEADING
"SALES OVER INVENTORY AND RETURNS"
"ACROSS PRODUCT CODE"
SUM SALES OVER INV OVER RETURNS ACROSS PROD_CODE
END
    
```



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 1063. For online graphs, FOCUS automatically sets the height of your graph based on the terminal dimensions.
- ❑ 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 1061.
- ❑ 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 1062.

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 1061).

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.

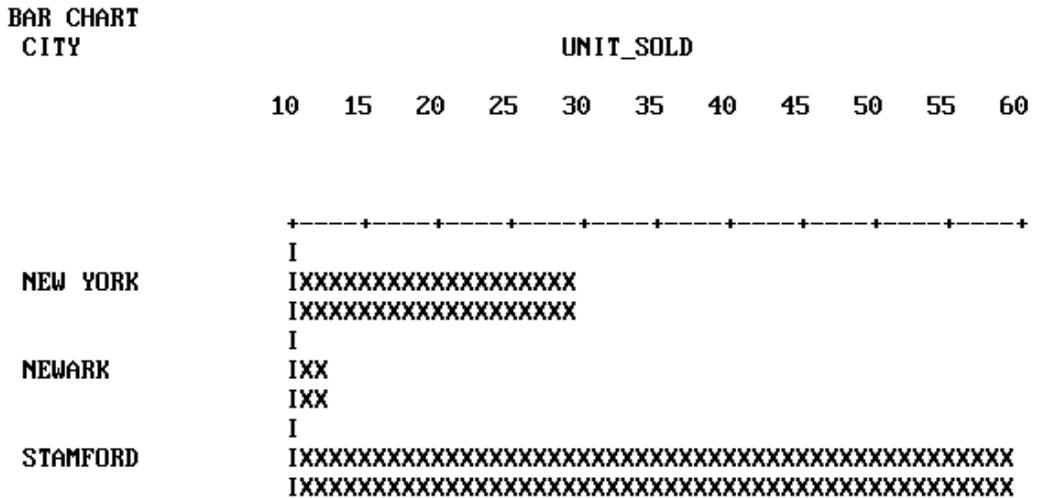
Bar Charts

In this section:
Bar Chart Features

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, BARSPEACE=1
```

```
GRAPH FILE SALES
HEADING
"BAR CHART"
SUM UNIT_SOLD BY CITY
IF PROD_CODE EQ B10
END
```



In the request above, the parameters BARSPEACE 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 BARSPEACE=2, BARWIDTH=2
```


Bar Chart Features

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
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END
```

```
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
```


- ❑ 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 1062.

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

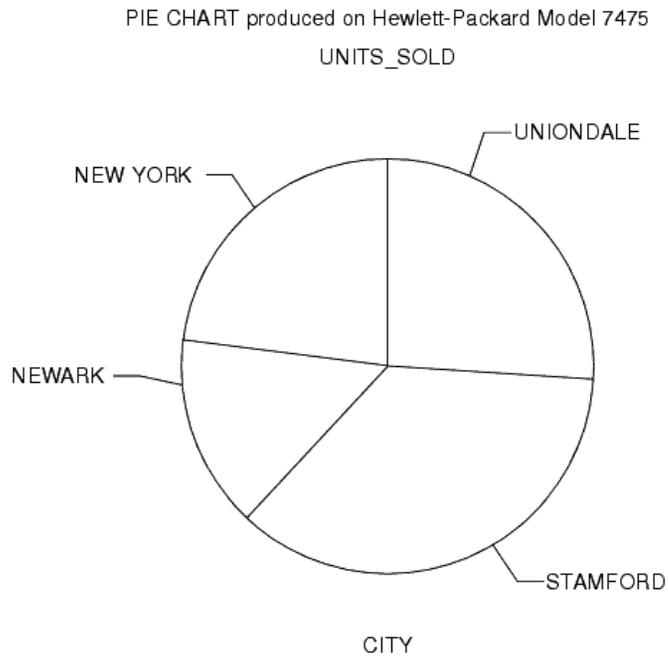
In this section:

Pie Chart Features

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 1072.

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
```

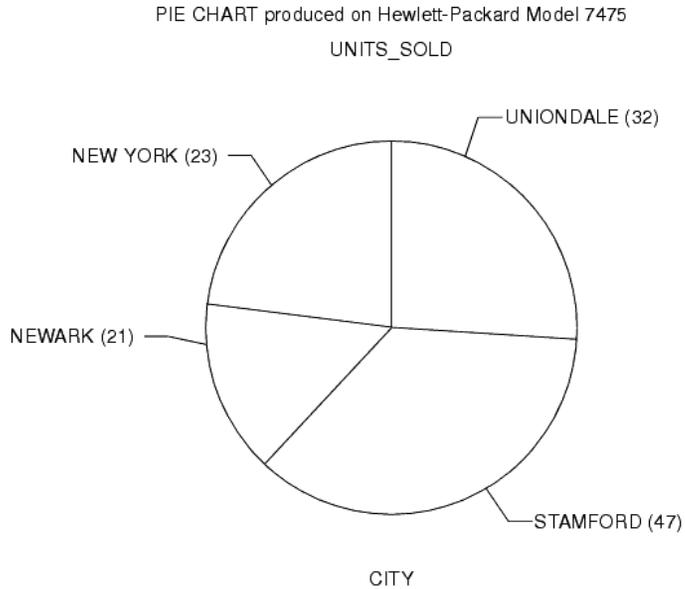


Pie Chart Features

To add summary numbers for each slice of the pie chart on the previous page, enter the following:

```
SET BARNUMB=ON  
REPLOTT
```

The effect is shown below:



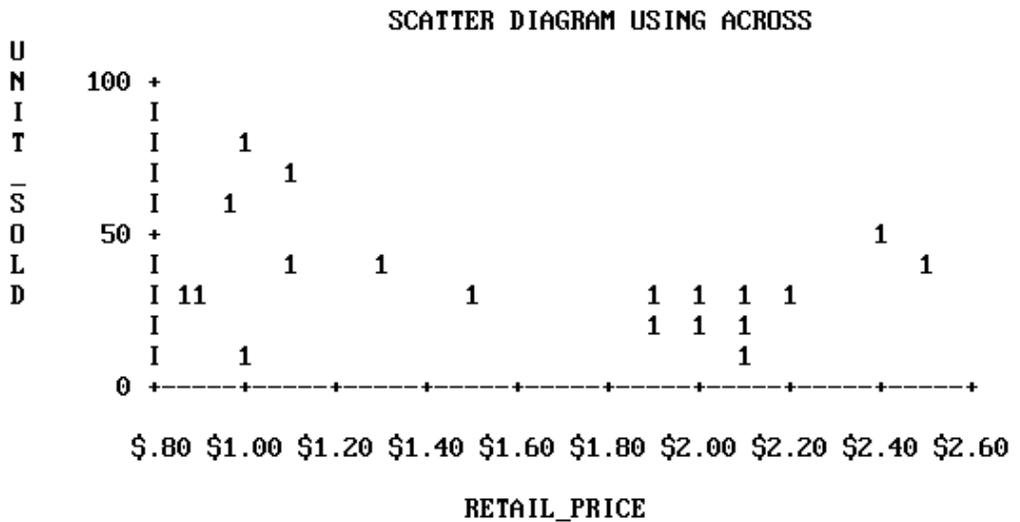
Note: FOCUS does not include a facility for displaying exploded pie chart slices.

Scatter Diagrams

In this section:
Scatter Diagram Features

Scatter diagrams illustrate occurrence patterns and distribution of variables. 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.

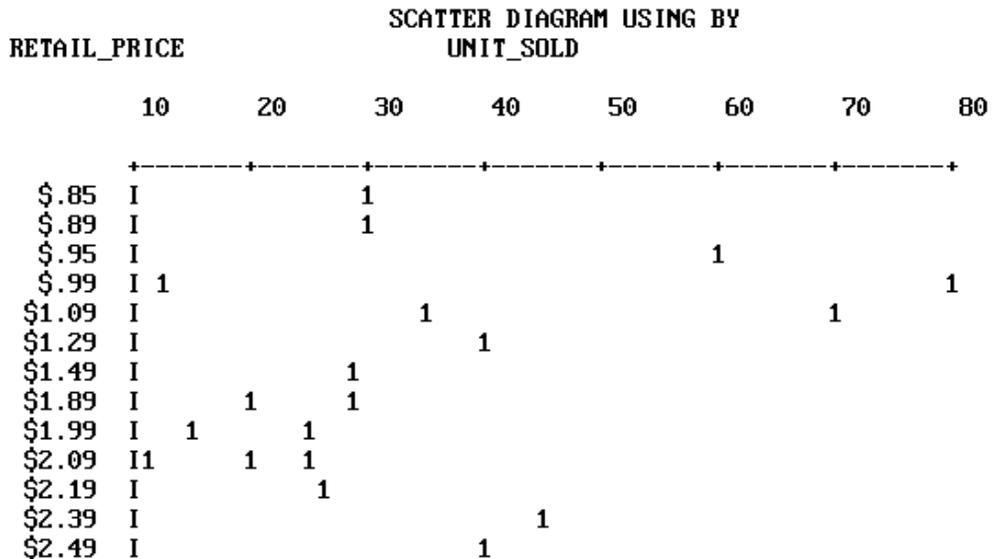
```
GRAPH FILE SALES
HEADING CENTER
"SCATTER DIAGRAM USING ACROSS"
PRINT UNIT_SOLD ACROSS RETAIL_PRICE
END
```



The point plots on the vertical axis represent the values for the ACROSS field named. Each record selected contributes a separate point. The sort control fields are plotted on the horizontal axis, which is also scaled if the control field values are numeric.

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 1060).

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

In this section:

The Horizontal Axis: System Defaults

The Vertical Axis: System Defaults

Highlighting Facilities

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 adding parallel lines across the horizontal and/or vertical axes, to form a grid against which the data is arrayed.

The following 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
GPROMPT	OFF
GRID	OFF
VGRID	OFF
HAXIS	130
VAXIS	66
GTREND	OFF
GRIBBON (GCOLOR)	OFF
VZERO	OFF
VAUTO	ON
VMAX	.00
VMIN	.00
HAUTO	ON
HMAX	.00
HMIN	.00
AUTOTICK	ON
HTICK	.00

Parameter	Setting
HCLASS	.00
VTICK	.00
VCLASS	.00
BARWIDTH	1
BARSPACE	0
BARNUMB	OFF
HISTOGRAM	ON
HSTACK	OFF
BSTACK	OFF
PIE	OFF
GMISSING	OFF
GMISSVAL	.00

For information about each of the parameters listed, refer to [SET Parameters](#) on page 1081.

The Horizontal Axis: System Defaults

How to:

Set the Width

Set the Scale: Assigning Fixed Limits

Set Class and Tick Intervals

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 1081.

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), turn off the automatic scaling mechanism, and set new limit values. This is done with the SET command. The syntax is:

```
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

How to:

Set the Height

Set the Scale: Assigning Fixed Limits

Set Class and Tick Intervals

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 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 1081).

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), turn off the automatic scaling mechanism, and set 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**How to:**

Add Horizontal or Vertical Grids

Add Summary Numbers in Pie and Bar Charts

Add Trend Lines on Scatter Plots

FOCUS contains the following facilities for highlighting the information shown on your graphs:

- ❑ 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:

```
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:

```
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

In this section:

- Plotting Dates
- Handling Missing Data
- Using Fixed-Axis Scales
- Saving Formatted GRAPH Output
- Creating Formatted Input for CA-TELLAGRAF
- Using the FOCUS ICU Interface

The following topics have general applicability for many graph applications:

- 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 are 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.

You may wish to selectively combine groups of date point plots to reduce the number of separate points on the horizontal axis. 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
```

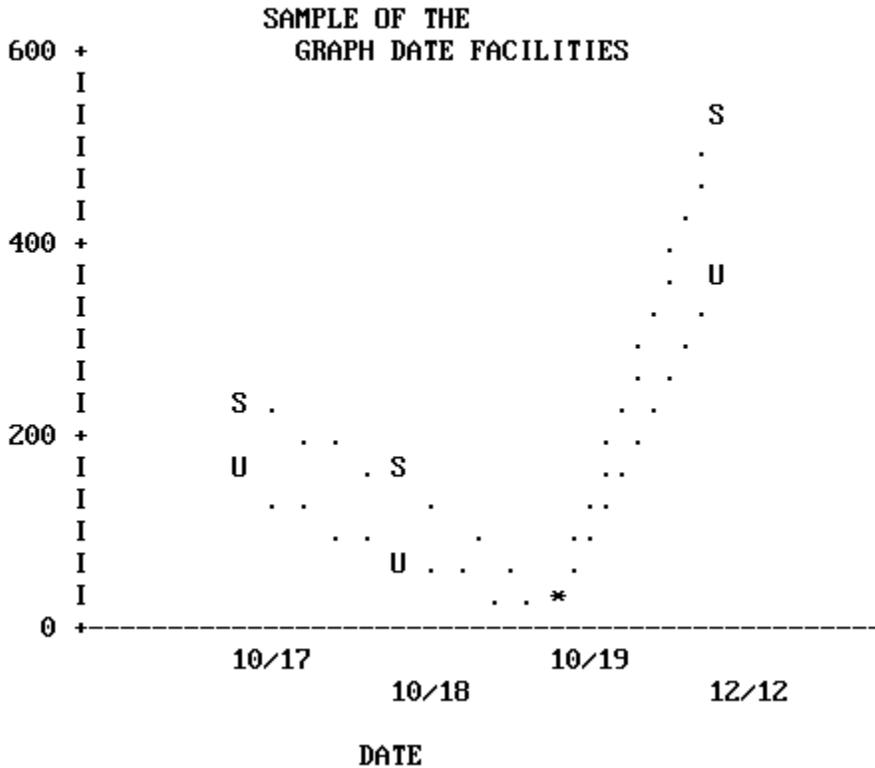
The example that follows illustrates a graph with date plots. Run it as an offline request:

```

SET HISTOGRAM=OFF
SET AUTOTICK=OFF, VCLASS = 200, VTICK = 25
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END
GRAPH FILE SALES
HEADING
"</6 <22 SAMPLE OF THE"
"<24 GRAPH DATE FACILITIES"
SUM SALES AND UNIT_SOLD ACROSS DATE
END
    
```

```

S SALES
U UNIT_SOLD
    
```



Handling Missing Data

You can handle missing data selectively in GRAPH requests. You can portray the missing data as null values, or 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.

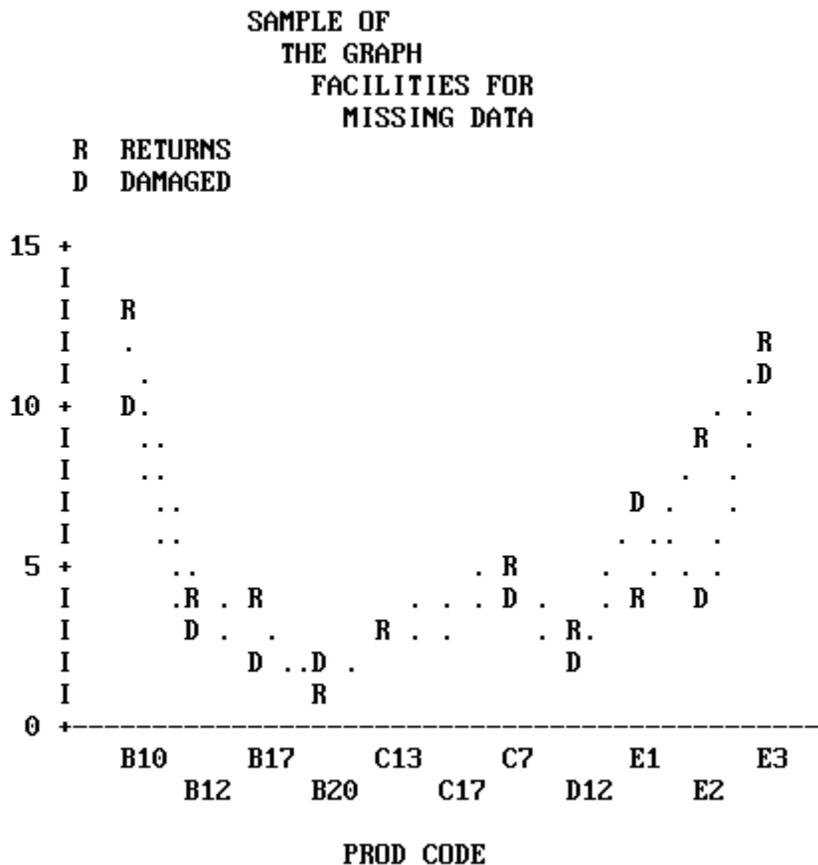
Normally, missing values on the vertical axis are ignored (VZERO=OFF). If ON, the values are treated as zero (0).

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 1081).

The examples that follow illustrate the same request, but with different treatments of missing values selected. Run them as offline requests:

```

SET GMISSING=ON, HIST=OFF
SET AUTOTICK=OFF, VCLASS=5, VTICK=1
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
    
```

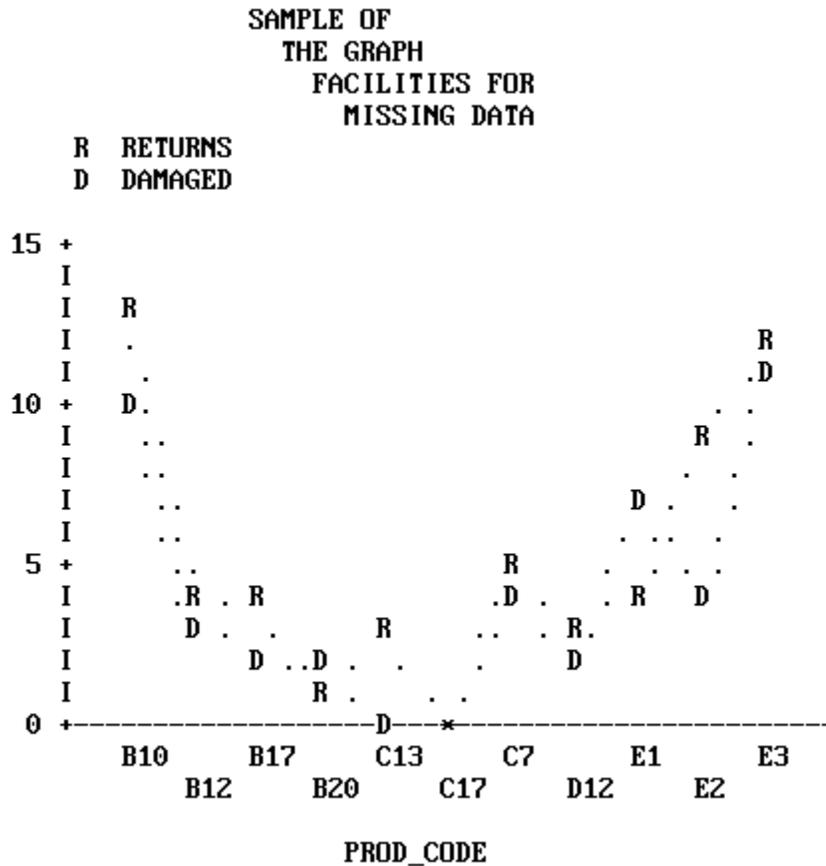


In this example GMISSING is ON and GMISSVAL is 0, so the graph ignores zero values for products C13 and C17.

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 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 [The Horizontal Axis: System Defaults](#) on page 1060, and [How to Set the Scale: Assigning Fixed Limits](#) on page 1061.

Saving Formatted GRAPH Output

How to:

Display Stored Graphs

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 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 1076). 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 FOCUSAVE.

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 1081), 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 allocates 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.

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 use ICU to generate graphs.

Directions for using this interface can be found in the *ICU Interface Users Manual*.

Special Graphics Devices

In this section:

Medium-Resolution Devices

High-Resolution 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.

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 1081), 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

In this section:

- IBM Devices Using GDDM
- GDDM Default Conditions
- GDDM Save and Print Facilities
- Graphics Device Characteristics
- Hewlett-Packard Plotters
- Tektronix Color Terminals

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; press 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 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 1081). FOCUS provides default lengths and scaling of axes, but these and the other graphic elements can be changed by adjusting SET parameters discussed in [Adjusting Graph Elements](#) on page 1058 and summarized in [SET Parameters](#) on page 1081.

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 1081.

Command and SET Parameter Summary

In this section:

GRAPH Command

SET Parameters

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 additional control at run time (for example, pause to adjust paper before printing, select a device, etc.).

GRAPH Command

How to:

- Enter the Environment
- Specify Annotating Text
- Name the Subject and Graph Type
- Specify Display Fields
- Specify Horizontal Sorting of Data Points
- Specify Separate Graphs or Vertical Sorting of Plot Points
- Save the Formatted Graph Data in a File
- Complete the GRAPH Request
- Concatenate Unlike Data Sources

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:

```
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:

```
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. Make sure, however, 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/OFF</u>	When ON, FOCUS automatically sets the tick mark intervals. (See also HTICK and VTICK.)
BARNUMB	<u>ON/OFF</u>	Places the summary values at the ends of the bars on bar charts, or slices on pie charts.
BARSPACE	0-20	Specifies the number of lines separating the bars on bar charts.
BARWIDTH	1-20	Specifies the number of lines per bar on bar charts.
BSTACK	<u>ON/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 TERMINAL	IBM3270	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).
Medium-resolution devices:		
	AJ	Specifies Anderson Jacobson - Model AJ830.
	AJ12	Specifies Anderson Jacobson - Model AJ832 (12 Pitch).
	DIABLO	Specifies Diablo - Model 1620.
	DIABLO12	Specifies Diablo - Model 1620 (12 Pitch).
	GS	Specifies Gencom.
	GS12	Specifies Gencom (12 Pitch).
	HIGHRES	Specifies generic device for most medium-resolution graphic devices.
	HIGHRS12	Specifies generic device -see above (12 Pitch).
	TRENDATA	Specifies Trendata - Model 4000A.
	TRENDT12	Specifies Trendata - Model 4000A (12 Pitch).
High-resolution devices from Hewlett-Packard:		
	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.

Parameter Name	Range of Values	Parameter Function
	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:
HAXIS=100, VAXIS=50.

Parameter Name	Range of Values	Parameter Function
High-resolution devices from IBM:		
	IBM3279	Specifies one of the following devices: 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. 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
High-resolution devices from Tektronix:		
	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.
	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 (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. Note: 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.
GMISSVAL	nn	Specifies the variable value that represents missing data.
GPROMPT	ON/OFF	When ON, FOCUS prompts for all pertinent graph parameters.
GRIBBON		See GCOLOR.
GRID	ON/OFF	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/OFF	When ON, specifies that a trend line is to appear on scatter diagrams.
HAUTO	ON/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- 130	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.

Parameter Name	Range of Values	Parameter Function
HCLASS	nnn	Specifies the horizontal class interval when AUTOTICK=OFF.
HISTOGRAM	ON/OFF	When ON, FOCUS draws a histogram instead of a curve when values on the horizontal axis are not numeric.
HMAX	nnn	Specifies the maximum value on the horizontal axis when automatic scaling is not used (HAUTO=OFF).
HMIN	nnn	Specifies the minimum value on the horizontal axis when automatic scaling is not used (HAUTO=OFF).
HSTACK	ON/OFF	Specifies that the bars on a histogram are to be stacked rather than placed side by side.
HTICK	nnn	Specifies the horizontal axis tick mark interval, when AUTOTICK is OFF.
PAUSE	ON/OFF	Specifies whether there is a pause for paper adjustment on the plotter after the request is executed.
PIE	ON/OFF	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> <pre>SET PLOT=0050,0018</pre> <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>

Parameter Name	Range of Values	Parameter Function
PRINT	ONLINE/OFFLINE	When OFFLINE is entered, the graph is printed on the system high-speed printer.
TERM		See DEVICE.
VAUTO	ON/OFF	Specifies automatic scaling of the vertical axis unless overridden by the user. If OFF, the user must supply values for VMAX and VMIN.
VAXIS	20-66	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	nnn	Specifies the vertical class interval when AUTOTICK=OFF.
VGRID	ON/OFF	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).
VMAX	nnn	Specifies the maximum value on the vertical axis when automatic scaling is not used (VAUTO=OFF).
VMIN	nnn	Specifies the minimum value on the vertical axis when automatic scaling is not used (VAUTO=OFF).
VTICK	nnn	Specifies the vertical axis tick mark interval, when AUTOTICK is OFF.
VZERO	ON/OFF	With VZERO=OFF, missing values on the vertical axis are ignored. If ON, the values are treated as zero (0).

22 | Using SQL to Create Reports

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.

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

Supported and Unsupported SQL Statements

Reference:

Supported SQL Statements
Unsupported SQL Statements
SQL Translator Reserved Words

SQL Translation Services is compliant with ANSI Level 2. This facility supports many, but not all, SQL statements. RDBMS engines may also support the *alpha1* CONCAT *alpha2* syntax. See [Supported SQL Statements](#) on page 1090 and [Unsupported SQL Statements](#) on page 1092.

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 1096.
- ❑ 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 that 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 Mainframe: 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. See [SQL Translator Support for Date, Time, and Timestamp Fields](#) on page 1104.
- ❑ 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 clauses expressed using explicit column names.
- ❑ ORDER BY clauses 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).

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.
- ❑ Temporary defined columns. Permanent defined columns, defined in the Master File, are supported.
- ❑ 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

In this section:

The SQL SELECT Statement

SQL Joins

SQL CREATE TABLE and INSERT INTO Commands

SQL CREATE VIEW and DROP VIEW Commands

SQL PREPARE, EXECUTE, and COMMIT Commands

Cartesian Product Style Answer Sets

Continental Decimal Notation (CDN)

Specifying Field Names in SQL Requests

SQL UNION, INTERSECT, and EXCEPT Operators

Numeric Constants, Literals, Expressions, and Functions

How to:

Use SQL Translator Commands

Reference:

TABLE Formatting Phrases in SQL Requests

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: use the JOIN command before you issue any SQL statements, or use the WHERE predicate in the SQL SELECT statement to join the required files dynamically. See [SQL Joins](#) on page 1096.

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. See [Supported SQL Statements](#) on page 1090.

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

Are optional TABLE formatting phrases. See [TABLE Formatting Phrases in SQL Requests](#) on page 1094.

END or QUIT

Is required to terminate the 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:

- ❑ Use TABLE formatting phrases with SELECT and UNION only.
- ❑ 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 ON TABLE HOLD or ON TABLE PCHOLD, see [Saving and Reusing Your Report Output](#) on page 421.

- ❑ 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.

The SQL SELECT Statement

The 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. See [Displaying Report Data](#) on page 45.

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. See [Saving and Reusing Your Report Output](#) on page 421.

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 [Selecting Records for Your Report](#) on page 157.

Only subqueries based on equality, when the WHERE expression is compared to a subquery by 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. See [Merging Data Sources](#) on page 877.

For related information, see [Supported SQL Statements](#) on page 1090 and [How to Use SQL Translator Commands](#) on page 1093.

SQL Joins

How to:

Create an Inner Join

Create an Outer Join

Reference:

Join Name Assignments From the SQL Translator

SQL Join Considerations

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 retrieved from which data sources.

Joined fields in the SQL WHERE predicate must be qualified if the names are not unique; specify them 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; specify them 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 the SQL Translator

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 a maximum of 16 joins). The first join has the AS name SQLJNM01, the second join is 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 statement, 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 the exact value requested (in this case, '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. See [Index Optimized Retrieval](#) on page 1109.

SQL CREATE TABLE and INSERT INTO Commands

Reference:

Usage Notes for CREATE TABLE and INSERT INTO Commands

SQL Translator supports the commands CREATE TABLE and INSERT INTO table:

- ❑ CREATE TABLE creates a new data source table. It only generates single-segment Master Files.
- ❑ INSERT INTO inserts 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 INTO Commands

- ❑ According to normal SQL data definition syntax, each CREATE TABLE or INSERT INTO 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 INTO commands, the data type FLOAT should be declared with a precision and used in an INSERT INTO command 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 shows a 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
```

Example: Inserting Values Expressed With an Exponent

To insert the values 100.00E01 and 200.00E01, specify:

```
SQL
CREATE TABLE T (A FLOAT(6), B CHAR(4))
END
SQL
INSERT INTO T (A,B) VALUES (1000.00,'1234');
END
SQL
INSERT INTO T (A,B) VALUES (2000.00,'4321');
END
```

SQL CREATE VIEW and DROP VIEW Commands

How to:

Create a View

A view is a transient object that inherits most of the characteristics of a table. Like a table, it is composed of rows and columns:

- ❑ CREATE VIEW creates views. Note that it does not put the view in the system catalog.
- ❑ DROP VIEW explicitly removes transient tables and views from the environment.

Tip: To use a view, issue a SELECT from it. 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 [Saving and Reusing Your Report Output](#) on page 421.

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
```

SQL PREPARE, EXECUTE, and COMMIT Commands

The SQL PREPARE statement creates a machine language form of an SQL statement and associates it with an identifier; you can then execute the SQL statement by referring to this identifier.

The COMMIT statement makes updates or inserts permanent, and clears all PREPARE statements.

Example: Executing an SQL SELECT Statement Using an Identifier

The following example executes an SQL SELECT statement by referencing the identifier PREP_QUERY:

```
SQL
  PREPARE PREP_QUERY FROM
  SELECT COUNTRY
  FROM CAR
  WHERE LENGTH > ? AND COUNTRY = ?;
END
SQL
  EXECUTE PREP_QUERY USING 165, 'ITALY';
END
SQL
  EXECUTE PREP_QUERY USING 190, 'ENGLAND';
END
SQL
  EXECUTE PREP_QUERY USING 182, 'FRANCE';
END
SQL
  COMMIT;
END
```

In the statement, each question mark (?) indicates where to substitute a value. Provide the necessary values in the USING clause of the EXECUTE statement in the order of the question marks in the original statement.

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 [Merging Data Sources](#) on page 877.

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

Specify fields in an SQL request 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.
- ❑ **Qualified field names.** 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 1096. See the *Describing Data With WebFOCUS Language* manual for more information.
- ❑ **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 as delimited identifiers, and not contained in the national character set of the installed FOCUS environment.

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:

```
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. See [Merging Data Sources](#) on page 877.

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 of these 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

In this section:

Extracting Date-Time Components Using the SQL Translator

Reference:

SQL Translator Support for Date, Time, and Timestamp Fields

Several new data types have been defined for 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, and 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 to 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 are 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

How to:

Use Date, Time, and Timestamp Functions Accepted by the SQL Translator
 Use the SQL Translator EXTRACT Function to Extract Date-Time Components

The SQL Translator supports several functions that return components from date-time values. Use the EXTRACT statement to extract components.

Use the TRIM function to remove leading and/or trailing patterns from date, time, and timestamp values. See the *Using Functions* manual.

Syntax: **How to Use 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
YEAR(<i>x</i>)	year
MONTH(<i>x</i>)	month number
DAY(<i>x</i>)	day number
HOUR(<i>x</i>)	hour
MINUTE(<i>x</i>)	minute
SECOND(<i>x</i>)	second
MILLISECOND(<i>x</i>)	millisecond
MICROSECOND(<i>x</i>)	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 VIDSQL 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=NOTTRUNC
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: **How to Use the SQL Translator EXTRACT Function to Extract Date-Time Components**

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

In this section:

Optimized Joins

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.

Optimized Joins

The SQL Translator accepts joins in SQL syntax. SQL language joins have no implied direction; the concepts of host and cross-referenced files do not exist in SQL.

The SQL Translator analyzes each join to identify efficient implementation. 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 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 requires changing the table order, the Translator abandons Phase 2 and implements Phase 3.
- 3.** The Translator generates the join structure from the lowest-cost joins first, and then from the more expensive joins as necessary. This sorting process may change the order in which tables are joined. The efficiency of the join that this procedure generates depends 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

How to:

Improve Performance Using SQLTOPTTF

To improve performance, the SQL Translator can be set to generate FOCUS TABLEF commands instead of TABLE commands. Take advantage of this optimization using the SET SQLTOPTTF command (SQL Translator OPTimization TableF). See [Improving Report Processing](#) on page 903.

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). ON is the default value.

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 introduces new rows into an existing table.
- ❑ The DELETE statement removes a row or combination of rows from a table.
- ❑ The UPDATE statement enables 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 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.

A Master Files and Diagrams

This appendix contains descriptions and structure diagrams for the sample data sources used throughout the documentation.

Topics:

- ❑ Creating Sample Data Sources
- ❑ EMPLOYEE Data Source
- ❑ JOBFIL Data Source
- ❑ EDUCFILE Data Source
- ❑ SALES Data Source
- ❑ PROD Data Source
- ❑ CAR Data Source
- ❑ LEDGER Data Source
- ❑ FINANCE Data Source
- ❑ REGION Data Source
- ❑ COURSES Data Source
- ❑ EMPDATA Data Source
- ❑ EXPERSON Data Source
- ❑ TRAINING Data Source
- ❑ COURSE Data Source
- ❑ JOBHIST Data Source
- ❑ JOBLIST Data Source
- ❑ LOCATOR Data Source
- ❑ PERSINFO Data Source
- ❑ SALHIST Data Source
- ❑ PAYHIST File
- ❑ COMASTER File
- ❑ VIDEOTRK, MOVIES, and ITEMS Data Sources
- ❑ VIDEOTR2 Data Source
- ❑ Gotham Grinds Data Sources
- ❑ Century Corp Data Sources

Creating Sample Data Sources

Create sample data sources on your user ID by executing the procedures specified below. These FOCXECs 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, EDUCFILE, and JOBFILE	<p>Under CMS, enter:</p> <p><code>EX EMPTEST</code></p> <p>Under MVS, enter:</p> <p><code>EX EMPTSO</code></p> <p>These FOCXECs 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 FOCXEC replaces them with new copies. This FOCXEC assumes that the high-level qualifier for the FOCUS data sources is the same as the high-level qualifier for the MASTER PDS that was unloaded from the tape.</p>
SALES PROD	<p><code>EX SALES</code></p> <p><code>EX PROD</code></p>
CAR	None (created automatically during installation).
LEDGER FINANCE REGION COURSES EXPERSON	<p><code>EX LEDGER</code></p> <p><code>EX FINANCE</code></p> <p><code>EX REGION</code></p> <p><code>EX COURSES</code></p> <p><code>EX EXPERSON</code></p>

Data Source	Load Procedure Name
EMPDATA TRAINING COURSE JOBHIST JOBLIST LOCATOR PERSINFO SALHIST	EX LOADPERS
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 MOVIES	EX LOADVTRK
VIDEOTR2	EX LOADVID2
Gotham Grinds	EX DBLGG
Century Corp: CENTCOMP CENTFIN CENTHR CENTINV CENTORD CENTQA CENTGL CENTSYSF CENTSTMT	EX LOADCOM EX LOADFIN EX LOADHR EX LOADINV EX LOADORD EX LOADCQA EX LDCENTGL EX LDCENTSY EX LDSTMT

EMPLOYEE Data Source

In this section:

EMPLOYEE Master File

EMPLOYEE Structure Diagram

EMPLOYEE contains sample data about company employees. Its segments are:

[EMPINFO](#)

Contains employee IDs, names, and positions.

[FUNDTRAN](#)

Specifies employee direct deposit accounts. This segment is unique.

[PAYINFO](#)

Contains the employee salary history.

[ADDRESS](#)

Contains employee home and bank addresses.

[SALINFO](#)

Contains data on employee monthly pay.

[DEDUCT](#)

Contains data on monthly pay deductions.

EMPLOYEE also contains cross-referenced segments belonging to the JOBFIL and EDUCFIL files, also described in this appendix. The segments are:

[JOBSEG \(from JOBFIL\)](#)

Describes the job positions held by each employee.

[SKILLSEG \(from JOBFIL\)](#)

Lists the skills required by each position.

SECSEG (from JOBFIL)

Specifies the security clearance needed for each job position.

ATTNDSEG (from EDUCFILE)

Lists the dates that employees attended in-house courses.

COURSEG (from EDUCFILE)

Lists the courses that the employees attended.

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=COURSEG, SEGTYPE=KLU, PARENT=ATTNDSEG, CRFILE=EDUCFILE,$

```


JOBFILE Data Source

In this section:

JOBFILE Master File

JOBFILE Structure Diagram

JOBFILE contains sample data about company 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.

JOBFILE Master File

```
FILENAME=JOBFILE, SUFFIX=FOC
SEGNAME=JOBSEG, SEGTYPE=S1
  FIELDNAME=JOBCODE, ALIAS=JC, FORMAT=A3, INDEX=I, $
  FIELDNAME=JOB_DESC, ALIAS=JD, FORMAT=A25, $
SEGNAME=SKILLSEG, SEGTYPE=S1, PARENT=JOBSEG
  FIELDNAME=SKILLS, ALIAS=, FORMAT=A4, $
  FIELDNAME=SKILL_DESC, ALIAS=SD, FORMAT=A30, $
SEGNAME=SECSEG, SEGTYPE=U, PARENT=JOBSEG
  FIELDNAME=SEC_CLEAR, ALIAS=SC, FORMAT=A6, $
```

JOBFILE Structure Diagram

```

SECTION 01
                STRUCTURE OF FOCUS      FILE JOBFILE ON 05/15/03 AT 14.40.06

                JOBSEG
01             S1
*****
*JOBCODE      **I
*JOB_DESC     **
*             **
*             **
*             **
*****
                I
                +-----+
                I             I
                I SECSEG     I SKILLSEG
02             I U           03             I S1
*****
*SEC_CLEAR    *             *SKILLS      **
*             *             *SKILL_DESC **
*             *             *             **
*             *             *             **
*             *             *             **
*****
                *****
                *****

```

EDUCFILE Data Source

In this section:

EDUCFILE Master File

EDUCFILE Structure Diagram

EDUCFILE contains sample data about company in-house courses. Its segments are:

COURSESEG

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.

EDUCFILE Master File

```
FILENAME=EDUCFILE, SUFFIX=FOC
SEGNAME=COURSEG, SEGTYPE=S1
  FIELDNAME=COURSE_CODE, ALIAS=CC, FORMAT=A6, $
  FIELDNAME=COURSE_NAME, ALIAS=CD, FORMAT=A30, $
SEGNAME=ATTNDSEG, SEGTYPE=SH2, PARENT=COURSEG
  FIELDNAME=DATE_ATTEND, ALIAS=DA, FORMAT=I6YMD, $
  FIELDNAME=EMP_ID, ALIAS=EID, FORMAT=A9, INDEX=I, $
```

EDUCFILE Structure Diagram

```
SECTION 01
STRUCTURE OF FOCUS FILE EDUCFILE ON 05/15/03 AT 14.45.44
```

```
          COURSEG
01      S1
*****
*COURSE_CODE **
*COURSE_NAME **
*           **
*           **
*           **
*****
          I
          I
          I
          I ATTNDSEG
02      I SH2
*****
*DATE_ATTEND **
*EMP_ID      **I
*           **
*           **
*           **
*****
          *****
```

SALES Data Source

In this section:

SALES Master File

SALES Structure Diagram

SALES contains sample data about a dairy company with an affiliated 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. The PROD_CODE field is indexed. The RETURNS and DAMAGED fields have the MISSING=ON attribute.

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=I,$
  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,$

```

SALES Structure Diagram

SECTION 01

STRUCTURE OF FOCUS

FILE SALES ON 05/15/03 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 **
*          **
*****
*****
```

PROD Data Source

In this section:

PROD Master File

PROD Structure Diagram

The PROD data source lists products sold by a dairy company. It consists of one segment, PRODUCT. The field PROD_CODE is indexed.

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, $
```

PROD Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE PROD   ON 05/15/03 AT 14.57.38
          PRODUCT
01        S1
*****
*PROD_CODE   **I
*PROD_NAME   **
*PACKAGE     **
*UNIT_COST   **
*            **
*****
*****
```

CAR Data Source

In this section:

CAR Master File

CAR Structure Diagram

CAR contains sample data about 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.

WARANT

Lists the type of warranty.

EQUIP

Lists standard equipment.

The aliases in the CAR Master File are specified without the ALIAS keyword.

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 , $

```

CAR Structure Diagram

```

SECTION 01          STRUCTURE OF FOCUS      FILE CAR      ON 04/06/07 AT 11.13.56

          ORIGIN
01          S1
*****
* COUNTRY          **I
*
*
*
*****
          I
          I
          I
          I CORP
02          I S1
*****
* CAR
*
*
*
*****
          I
          I
          I
          I
          I CARREC          I WARRANT          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          *
*
*****

```

LEDGER Data Source

In this section:

LEDGER Master File

LEDGER Structure Diagram

LEDGER contains sample accounting data. 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.

LEDGER Master File

```
FILENAME=LEDGER, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
FIELDNAME=YEAR, , FORMAT=A4, $
FIELDNAME=ACCOUNT, , FORMAT=A4, $
FIELDNAME=AMOUNT, , FORMAT=I5C,$
```

LEDGER Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE LEDGER   ON 05/15/03 AT 15.17.08

          TOP
01       S2
*****
*YEAR           **
*ACCOUNT        **
*AMOUNT         **
*               **
*               **
*****
*****
```

FINANCE Data Source

In this section:

FINANCE Master File

FINANCE Structure Diagram

FINANCE contains sample financial data 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.

FINANCE Master File

```
FILENAME=FINANCE, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
FIELDNAME=YEAR, , FORMAT=A4, $
FIELDNAME=ACCOUNT, , FORMAT=A4, $
FIELDNAME=AMOUNT, , FORMAT=D12C,$
```

FINANCE Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE FINANCE   ON 05/15/03 AT 15.17.08

          TOP
01      S2
*****
*YEAR          **
*ACCOUNT       **
*AMOUNT        **
*              **
*              **
*****
*****
```

REGION Data Source

In this section:

REGION Master File

REGION Structure Diagram

REGION contains sample account data for the eastern and western 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.

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,$
```

REGION Structure Diagram

```
SECTION 01
STRUCTURE OF FOCUS FILE REGION ON 05/15/03 AT 15.18.48

TOP
01 S1
*****
*ACCOUNT **
*E_ACTUAL **
*E_BUDGET **
*W_ACTUAL **
* **
*****
*****
```

COURSES Data Source

In this section:

COURSES Master File

COURSES Structure Diagram

COURSES contains sample data about education courses. It consists of one segment, CRSESEG1. The field DESCRIPTION has a format of TEXT (TX).

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, $
```

COURSES Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE COURSES   ON 05/15/03 AT 12.26.05

          CRSESEG1
01      S1
*****
*COURSE_CODE **I
*COURSE_NAME **
*DURATION    **
*DESCRIPTION **T
*           **
*****
*****
```

EMPDATA Data Source

In this section:

EMPDATA Master File

EMPDATA Structure Diagram

EMPDATA contains sample data about company employees. It consists of one segment, EMPDATA. The PIN field is indexed. The AREA field is a temporary field.

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=DIV,          ALIAS=CDIV,         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 DIV (NE 'NORTH EASTERN' SE 'SOUTH EASTERN'
CE 'CENTRAL' WE 'WESTERN' CORP 'CORPORATE' ELSE 'INVALID AREA');$

```

EMPDATA Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS      FILE EMPDATA   ON 05/15/03 AT 14.49.09

          EMPDATA
01      S1
*****
*PIN          **I
*LASTNAME     **
*FIRSTNAME    **
*MIDINITIAL   **
*             **
*****
*****

```

EXPERSON Data Source

In this section:

EXPERSON Master File

EXPERSON Structure Diagram

The EXPERSON data source contains personal data about individual employees. It consists of one segment, ONESEG.

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=YEAR        , 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    , $

```

EXPERSON Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE EXPERSO  ON 05/15/03 AT 14.50.58

          ONESEG
01      S1
*****
*SOC_SEC_NO  **
*FIRST_NAME  **
*LAST_NAME   **
*AGE         **
*           **
*****
*****
```

TRAINING Data Source

In this section:

- TRAINING Master File
- TRAINING Structure Diagram

TRAINING contains sample data about training courses for employees. It consists of one segment, TRAINING. The PIN field is indexed. The EXPENSES, GRADE, and LOCATION fields have the MISSING=ON attribute.

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, $
```

TRAINING Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS      FILE TRAINING ON 05/15/03 AT 14.51.28

          TRAINING
01      SH3
*****
*PIN          **I
*COURSESTART **
*COURSECODE   **
*EXPENSES     **
*             **
*****
*****
```

COURSE Data Source

In this section:

- COURSE Master File
- COURSE Structure Diagram

COURSE contains sample data about education courses. It consists of one segment, CRSELIST.

COURSE Master File

```
FILENAME=COURSE, SUFFIX=FOC
SEGNAME=CRSELIST, SEGTYPE=S1
FIELDNAME=COURSECODE, ALIAS=CCOD, FORMAT=A7, INDEX=I, $
FIELDNAME=CTITLE, ALIAS=COURSE, FORMAT=A35, $
FIELDNAME=SOURCE, ALIAS=ORG, FORMAT=A35, $
FIELDNAME=CLASSIF, ALIAS=CLASS, FORMAT=A10, $
FIELDNAME=TUITION, ALIAS=FEE, FORMAT=D8.2, MISSING=ON, $
FIELDNAME=DURATION, ALIAS=DAYS, FORMAT=A3, MISSING=ON, $
FIELDNAME=DESCRIPTN1, ALIAS=DESC1, FORMAT=A40, $
FIELDNAME=DESCRIPTN2, ALIAS=DESC2, FORMAT=A40, $
FIELDNAME=DESCRIPTN2, ALIAS=DESC3, FORMAT=A40, $
```

COURSE Structure Diagram

```
SECTION 01
                STRUCTURE OF FOCUS   FILE COURSE   ON 05/15/03 AT 12.26.05

                CRSELIST
01             S1
*****
*COURSECODE   **I
*CTITLE       **
*SOURCE       **
*CLASSIF     **
*             **
*****
*****
```

JOBHIST Data Source

In this section:

- JOBHIST Master File
- JOBHIST Structure Diagram

JOBHIST contains information about employee jobs. Both the PIN and JOBSTART fields are keys. The PIN field is indexed.

JOBHIST Master File

```
FILENAME=JOBHIST, SUFFIX=FOC
SEGNAME=JOBHIST, SEGTYPE=SH2
FIELDNAME=PIN,      ALIAS=ID,      FORMAT=A9,      INDEX=I , $
FIELDNAME=JOBSTART, ALIAS=SDAT,    FORMAT=YMD,      $
FIELDNAME=JOBCLASS, ALIAS=JCLASS,   FORMAT=A8,      $
FIELDNAME=FUNCTITLE, ALIAS=FUNC,     FORMAT=A20,     $
```

JOBHIST Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS      FILE JOBHIST  ON 01/22/08 AT 16.23.46
          JOBHIST
          01      SH2
          *****
          *PIN          **I
          *JOBSTART    **
          *JOBCLASS    **
          *FUNCTITLE   **
          *              **
          *****
          *****
```

JOBLIST Data Source

In this section:

- JOBLIST Master File
- JOBLIST Structure Diagram

JOBLIST contains information about jobs. The JOBCLASS field is indexed.

JOBLIST Master File

```
FILENAME=JOBLIST, SUFFIX=FOC
SEGNAME=JOBSEG,  SEGTYPE=S1
FIELDNAME=JOBCLASS, ALIAS=JCLASS,  FORMAT=A8, INDEX=I , $
FIELDNAME=CATEGORY, ALIAS=JGROUP,  FORMAT=A25, $
FIELDNAME=JOBDESC, ALIAS=JDESC,  FORMAT=A40, $
FIELDNAME=LOWSAL, ALIAS=LSAL,  FORMAT=D12.2M, $
FIELDNAME=HIGHSAL, ALIAS=HSAL,  FORMAT=D12.2M, $
DEFINE GRADE/A2=EDIT (JCLASS,'$$$99');$
DEFINE LEVEL/A25=DECODE GRADE (08 'GRADE 8' 09 'GRADE 9' 10
'GRADE 10' 11 'GRADE 11' 12 'GRADE 12' 13 'GRADE 13' 14 'GRADE 14');$
```

JOBLIST Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE JOBLIST   ON 01/22/08 AT 16.24.52
          JOBSEG
          01      S1
          *****
          *JOBCLASS      **I
          *CATEGORY      **
          *JOBDESC       **
          *LOWSAL        **
          *               **
          *****
          *****
```

LOCATOR Data Source

In this section:

LOCATOR Master File

LOCATOR Structure Diagram

JOBHIST contains information about employee location and phone number. The PIN field is indexed.

LOCATOR Master File

```
FILENAME=LOCATOR, SUFFIX=FOC
SEGNAME=LOCATOR,  SEGTYPE=S1,
FIELDNAME=PIN,    ALIAS=ID_NO,    FORMAT=A9,    INDEX=I,    $
FIELDNAME=SITE,   ALIAS=SITE,    FORMAT=A25,   $
FIELDNAME=FLOOR,  ALIAS=FL,     FORMAT=A3,    $
FIELDNAME=ZONE,   ALIAS=ZONE,   FORMAT=A2,    $
FIELDNAME=BUS_PHONE, ALIAS=BTEL,   FORMAT=A5,    $
```

LOCATOR Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE LOCATOR   ON 01/22/08 AT 16.26.55
          LOCATOR
          01      S1
          *****
          *PIN          **I
          *SITE         **
          *FLOOR        **
          *ZONE         **
          *              **
          *****
          *****
```

PERSINFO Data Source

In this section:

- PERSINFO Master File
- PERSINFO Structure Diagram

PERSINFO contains employee personal information. The PIN field is indexed.

PERSINFO Master File

```
FILENAME=PERSINFO, SUFFIX=FOC
SEGNAME=PERSONAL, SEGTYPE=S1
FIELDNAME=PIN, ALIAS=ID, FORMAT=A9, INDEX=I, $
FIELDNAME=INCAREOF, ALIAS=ICO, FORMAT=A35, $
FIELDNAME=STREETNO, ALIAS=STR, FORMAT=A20, $
FIELDNAME=APT, ALIAS=APT, FORMAT=A4, $
FIELDNAME=CITY, ALIAS=CITY, FORMAT=A20, $
FIELDNAME=STATE, ALIAS=PROV, FORMAT=A4, $
FIELDNAME=POSTALCODE, ALIAS=ZIP, FORMAT=A10, $
FIELDNAME=COUNTRY, ALIAS=CTRY, FORMAT=A15, $
FIELDNAME=HOMEPHONE, ALIAS=TEL, FORMAT=A10, $
FIELDNAME=EMERGENCYNO, ALIAS=ENO, FORMAT=A10, $
FIELDNAME=EMERGCONTACT, ALIAS=ENAME, FORMAT=A35, $
FIELDNAME=RELATIONSHIP, ALIAS=REL, FORMAT=A8, $
FIELDNAME=BIRTHDATE, ALIAS=BDAT, FORMAT=YMD, $
```

PERSINFO Structure Diagram

```
SECTION 01
                STRUCTURE OF FOCUS      FILE PERSINFO ON 01/22/08 AT 16.27.24
                PERSONAL
01             S1
*****
*PIN           **I
*INCAREOF     **
*STREETNO     **
*APT          **
*             **
*****
*****
```

SALHIST Data Source

In this section:

- SALHIST Master File
- SALHIST Structure Diagram

SALHIST contains information about employee salary history. The PIN field is indexed. Both the PIN and EFFECTDATE fields are keys.

SALHIST Master File

```
FILENAME=SALHIST,   SUFFIX=FOC
SEGNAME=SLHISTRY,  SEGTYPE=SH2
FIELDNAME=PIN,     ALIAS=ID,      FORMAT=A9,        INDEX=I,         $
FIELDNAME=EFFECTDATE, ALIAS=EDAT,   FORMAT=YMD,        $
FIELDNAME=OLDSALARY, ALIAS=OSAL,   FORMAT=D12.2,     $
```

SALHIST Structure Diagram

```
SECTION 01
                STRUCTURE OF FOCUS      FILE SALHIST ON 01/22/08 AT 16.28.02
                SLHISTRY
01             SH2
*****
*PIN           **I
*EFFECTDATE    **
*OLDSALARY     **
*             **
*             **
*****
*****
```

PAYHIST File

In this section:

- PAYHIST Master File
- PAYHIST Structure Diagram

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.

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,$
```

PAYHIST Structure Diagram

```
SECTION 01
STRUCTURE OF FIX      FILE PAYHIST ON 05/15/03 AT 14.51.59

      PAYSEG
01      S1
*****
*SOC_SEC_NO  **
*DATE_OF_IN  **
*AMT_OF_INC  **
 *PCT_INC    **
 *           **
*****
*****
```

COMASTER File

In this section:

COMASTER Master File
COMASTER Structure Diagram

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, which lists file information.
- ❑ RECID, which lists segment information.
- ❑ FIELDID, which lists field information.
- ❑ DEFREC, which lists a description record.
- ❑ PASSREC, which lists read/write access.
- ❑ CRSEG, which lists cross-reference information for segments.
- ❑ ACCSEG, which lists DBA information.

COMASTER Master File

```

SUFFIX=COM , SEGNAME=FILEID
  FIELDNAME=FILENAME      ,FILE      ,A8 ,      , $
  FIELDNAME=FILE SUFFIX  ,SUFFIX   ,A8 ,      , $
  FIELDNAME=FDFCENT      ,DFC      ,A4 ,      , $
  FIELDNAME=FYRTHRESH    ,FYRT     ,A2 ,      , $
SEGNAME=RECID
  FIELDNAME=SEGNAME      ,SEGMENT  ,A8 ,      , $
  FIELDNAME=SEGTYPE      ,SEGTYPE  ,A4 ,      , $
  FIELDNAME=SEGSIZE      ,SEGSIZE  ,I4 ,      A4, $
  FIELDNAME=PARENT       ,PARENT   ,A8 ,      , $
  FIELDNAME=CRKEY        ,VKEY     ,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 ,     , $
  FIELDNAME=DFCENT       ,DFC      ,A4 ,      , $
  FIELDNAME=YRTHRESH     ,YRT      ,A4 ,      , $
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=DBA FILE     ,          ,A8 ,      , $
  FIELDNAME=USER         ,PASS     ,A8 ,      , $
  FIELDNAME=ACCESS       ,ACCESS   ,A8 ,      , $
  FIELDNAME=RESTRICT     ,RESTRICT ,A8 ,      , $
  FIELDNAME=NAME         ,NAME     ,A66 ,     , $
  FIELDNAME=VALUE        ,VALUE    ,A80 ,     , $

```

COMASTER Structure Diagram

SECTION 01

STRUCTURE OF EXTERNAL FILE COMASTER ON 05/15/03 AT 14.53.38

```

      FILEID
01      SO
*****
*FILENAME **
*FILE SUFFIX **
*DEFRCENT **
*PERTHRESH **
*
*****
      I
      +-----+
      I          I
02      I RECID      07      I PASSREC
      I N          I N
*****
*SEGNAME **      *READ/WRITE **
*SEGTYPE **      *
*SEGSIZE **      *
*PARENT **      *
*
*****
      I
      +-----+
      I          I
03      I FIELDID   06      I CRSRG
      I N          I N
*****
*FIELDNAME **    *CRFILENAME **
*ALIAS **        *CRSEGNAME **
*FORMAT **      *ENCRYPT **
*ACTUAL **      *
*
*****
      I
      I
      I
      I DEFRC
04      I N
*****
*DEFINITION **
*
*
*
*****
      I
      I
      I
      I ACCSRG
05      I N
*****
*DBA **
*DBAFILE **
*USER **
*ACCESS **
*
*****

```

VIDEOTRK, MOVIES, and ITEMS Data Sources

In this section:

- VIDEOTRK Master File
- VIDEOTRK Structure Diagram
- MOVIES Master File
- MOVIES Structure Diagram
- ITEMS Master File
- ITEMS Structure Diagram

VIDEOTRK contains sample data about customer, rental, and purchase information for a video rental business. It can be joined to the MOVIES or ITEMS 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, $

```

VIDEOTRK Structure Diagram

SECTION 01

STRUCTURE OF FOCUS

FILE VIDEOTRK ON 05/15/03 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 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,   $
```

MOVIES Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS      FILE MOVIES      ON 05/15/03 AT 12.26.05

          MOVINFO
01          S1
*****
*MOVIECODE  **I
*TITLE      **
*CATEGORY   **
*DIRECTOR   **
*           **
*****
*****
```

ITEMS Master File

```

FILENAME=ITEMS,    SUFFIX=FOC
SEGNAME=ITMINFO,  SEGTYPE=S1
  FIELDNAME=PRODCODE,  ALIAS=PCOD,  FORMAT=A6,  INDEX=I,  $
  FIELDNAME=PRODNAME,  ALIAS=PROD,  FORMAT=A20,  $
  FIELDNAME=OURCOST,   ALIAS=WCost,  FORMAT=F6.2,  $
  FIELDNAME=RETAILPR,  ALIAS=PRICE,  FORMAT=F6.2,  $
  FIELDNAME=ON_HAND,   ALIAS=NUM,    FORMAT=I5,    $

```

ITEMS Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS      FILE ITEMS      ON 05/15/03 AT 12.26.05

```

```

          ITMINFO
01      S1
*****
*PRODCODE      **I
*PRODNAME      **
*OURCOST       **
*RETAILPR      **
*              **
*****
*****

```

VIDEOTR2 Data Source

In this section:

- VIDEOTR2 Master File
- VIDEOTR2 Structure Diagram

VIDEOTR2 contains sample data about customer, rental, and purchase information for a video rental business. It consists of four segments.

VIDEOTR2 Master File

```

FILENAME=VIDEOTR2, 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, $
  FIELDNAME=EMAIL, ALIAS=EMAIL, FORMAT=A18, $
SEGNAME=TRANSDAT, SEGTYPE=SH1, PARENT=CUST
  FIELDNAME=TRANSDATE, ALIAS=OUTDATE, FORMAT=HYMDI, $
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, $

```

VIDEOTR2 Structure Diagram

SECTION 01

STRUCTURE OF FOCUS

FILE VIDEOTR2 ON 05/15/03 AT 16.45.48

```

          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
*****
*TRANSCODE   **      *MOVIECODE   **I
*QUANTITY    **      *COPY          **
*TRANSTOT    **      *RETURNDATE  **
*            **      *FEE          **
*            **      *            **
*****
          *            **
          *****
          *****

```

Gotham Grinds Data Sources

In this section:

GGDEMOG Master File
GGDEMOG Structure Diagram
GGORDER Master File
GGORDER Structure Diagram
GGPRODS Master File
GGPRODS Structure Diagram
GGSALES Master File
GGSALES Structure Diagram
GGSTORES Master File
GGSTORES Structure Diagram

Gotham Grinds is a group of data sources that contain sample data about a specialty items company.

- ❑ GGDEMOG 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.
- ❑ GGORDER contains order information for Gotham Grinds. It consists of two segments, ORDER01 and ORDER02.
- ❑ GGPRODS contains product information for Gotham Grinds. It consists of one segment, PRODS01.
- ❑ GGSALES contains sales information for Gotham Grinds. It consists of one segment, SALES01.
- ❑ GGSTORES contains information for each of Gotham Grinds 12 stores in the United States. It consists of one segment, STORES01.

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=AVGHHSZ98, 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=AVGHHI98, 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=P15TO1998, ALIAS=E09, FORMAT=I09, TITLE='15 to 19',
DESC='Population 15 to 19 years old', $
FIELD=P20TO2998, ALIAS=E10, FORMAT=I09, TITLE='20 to 29',
DESC='Population 20 to 29 years old', $
FIELD=P30TO4998, ALIAS=E11, FORMAT=I09, TITLE='30 to 49',
DESC='Population 30 to 49 years old', $
FIELD=P50TO6498, 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', $

```

GGDEMOG Structure Diagram

```

SECTION 01
STRUCTURE OF FOCUS FILE GGDEMOG ON 05/15/03 AT 12.26.05

GGDEMOG
01 S1
*****
*ST **I
*HH **
*AVGHHSZ98 **
*MEDHHI98 **
* **
*****
*****

```

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 , $
    
```

GGORDER Structure Diagram

```

SECTION 01
                STRUCTURE OF FOCUS      FILE GGORDER  ON 05/15/03 AT 16.45.48

                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  :
:              :
:.....:
    
```

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=VCD, 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', $

```

GGPRODS Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS      FILE GGPRODS   ON 05/15/03 AT 12.26.05

          GGPRODS
01          S1
*****
*PRODUCT_ID  **I
*PRODUCT_DESC**I
*VENDOR_CODE **
*VENDOR_NAME **
*
*
*****
*****

```

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', $

```

GGSALES Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS      FILE GGSALES  ON 05/15/03 AT 12.26.05

          GGSALES
01        S1
*****
*SEQ_NO      **
*CATEGORY    **I
*PCD         **I
*PRODUCT     **I
*           **
*****
*****

```

GGSTORES Master File

```

FILENAME=GGSTORES, SUFFIX=FOC
SEGNAME=STORES01, SEGTYPE=S1
  FIELD=STORE_CODE, ALIAS=E02, FORMAT=A05, INDEX=I, 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=I, TITLE='State',
    DESC='State', $
  FIELD=ZIP, ALIAS=E08, FORMAT=A06, TITLE='Zip Code',
    DESC='Postal Code', $

```

GGSTORES Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS      FILE GGSTORES ON 05/15/03 AT 12.26.05

          GGSTORES
01          S1
*****
*STORE_CODE  **I
*STORE_NAME  **
*ADDRESS1    **
*ADDRESS2    **
*           **
*****
*****

```

Century Corp Data Sources

In this section:

CENTCOMP Master File
CENTCOMP Structure Diagram
CENTFIN Master File
CENTFIN Structure Diagram
CENTHR Master File
CENTHR Structure Diagram
CENTINV Master File
CENTINV Structure Diagram
CENTORD Master File
CENTORD Structure Diagram
CENTQA Master File
CENTQA Structure Diagram
CENTGL Master File
CENTGL Structure Diagram
CENTSYSF Master File
CENTSYSF Structure Diagram
CENTSTMT Master File
CENTSTMT Structure Diagram

Century Corp is a consumer electronics manufacturer that distributes products through retailers around the world. Century Corp has thousands of employees in plants, warehouses, and offices worldwide. Their mission is to provide quality products and services to their customers.

Century Corp is a group of data sources that contain financial, human resources, inventory, and order information. The last three data sources are designed to be used with chart of accounts data.

- ❑ CENTCOMP Master File contains location information for stores. It consists of one segment, COMPINFO.
- ❑ CENTFIN Master File contains financial information. It consists of one segment, ROOT_SEG.
- ❑ CENTHR Master File contains human resources information. It consists of one segment, EMPSEG.
- ❑ CENTINV Master File contains inventory information. It consists of one segment, INVINFO.
- ❑ CENTORD Master File contains order information. It consists of four segments, OINFO, STOSEG, PINFO, and INVSEG.
- ❑ CENTQA Master File contains problem information. It consists of three segments, PROD_SEG, INVSEG, and PROB_SEG.
- ❑ CENTGL Master File contains a chart of accounts hierarchy. The field GL_ACCOUNT_PARENT is the parent field in the hierarchy. The field GL_ACCOUNT is the hierarchy field. The field GL_ACCOUNT_CAPTION can be used as the descriptive caption for the hierarchy field.
- ❑ CENTSYSF Master File contains detail-level financial data. CENTSYSF uses a different account line system (SYS_ACCOUNT), which can be joined to the SYS_ACCOUNT field in CENTGL. Data uses "natural" signs (expenses are positive, revenue negative).
- ❑ CENTSTMT Master File contains detail-level financial data and a cross-reference to the CENTGL data source.

CENTCOMP Master File

```

FILE=CENTCOMP, SUFFIX=FOC, FDFC=19, FYRT=00
SEGNAME=COMPINFO, SEGTYPE=S1, $
FIELD=STORE_CODE, ALIAS=SNUM, FORMAT=A6, INDEX=I,
  TITLE='Store Id#:',
  DESCRIPTION='Store Id#', $
FIELD=STORENAME, ALIAS=SNAME, FORMAT=A20,
  WITHIN=STATE,
  TITLE='Store,Name:',
  DESCRIPTION='Store Name', $
FIELD=STATE, ALIAS=STATE, FORMAT=A2,
  WITHIN=PLANT,
  TITLE='State:',
  DESCRIPTION=State, $
DEFINE REGION/A5=DECODE STATE ('AL' 'SOUTH' 'AK' 'WEST' 'AR' 'SOUTH'
'AZ' 'WEST' 'CA' 'WEST' 'CO' 'WEST' 'CT' 'EAST'
'DE' 'EAST' 'DC' 'EAST' 'FL' 'SOUTH' 'GA' 'SOUTH' 'HI' 'WEST'
'ID' 'WEST' 'IL' 'NORTH' 'IN' 'NORTH' 'IA' 'NORTH'
'KS' 'NORTH' 'KY' 'SOUTH' 'LA' 'SOUTH' 'ME' 'EAST' 'MD' 'EAST'
'MA' 'EAST' 'MI' 'NORTH' 'MN' 'NORTH' 'MS' 'SOUTH' 'MT' 'WEST'
'MO' 'SOUTH' 'NE' 'WEST' 'NV' 'WEST' 'NH' 'EAST' 'NJ' 'EAST'
'NM' 'WEST' 'NY' 'EAST' 'NC' 'SOUTH' 'ND' 'NORTH' 'OH' 'NORTH'
'OK' 'SOUTH' 'OR' 'WEST' 'PA' 'EAST' 'RI' 'EAST' 'SC' 'SOUTH'
'SD' 'NORTH' 'TN' 'SOUTH' 'TX' 'SOUTH' 'UT' 'WEST' 'VT' 'EAST'
'VA' 'SOUTH' 'WA' 'WEST' 'WV' 'SOUTH' 'WI' 'NORTH' 'WY' 'WEST'
'NA' 'NORTH' 'ON' 'NORTH' ELSE ' ');,
  TITLE='Region:',
  DESCRIPTION=Region, $

```

CENTCOMP Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS          FILE CENTCOMP ON 05/15/03 AT 10.20.49

          COMPINFO
01      S1
*****
*STORE_CODE  **I
*STORENAME   **
*STATE       **
*            **
*            **
*****
*****

```

CENTFIN Master File

```

FILE=CENTFIN, SUFFIX=FOC, FDFC=19, FYRT=00
  SEGNAME=ROOT_SEG, SEGTYPE=S4, $
  FIELD=YEAR, ALIAS=YEAR, FORMAT=YY,
    WITHIN='*Time Period', $
  FIELD=QUARTER, ALIAS=QTR, FORMAT=Q,
    WITHIN=YEAR,
    TITLE=Quarter,
    DESCRIPTION=Quarter, $
  FIELD=MONTH, ALIAS=MONTH, FORMAT=M,
    TITLE=Month,
    DESCRIPTION=Month, $
  FIELD=ITEM, ALIAS=ITEM, FORMAT=A20,
    TITLE=Item,
    DESCRIPTION=Item, $
  FIELD=VALUE, ALIAS=VALUE, FORMAT=D12.2,
    TITLE=Value,
    DESCRIPTION=Value, $
  DEFINE ITYPE/A12=IF EDIT(ITEM,'9$$$$$$$$$$$$$$$$') EQ 'E'
    THEN 'Expense' ELSE IF EDIT(ITEM,'9$$$$$$$$$$$$$$$$') EQ 'R'
    THEN 'Revenue' ELSE 'Asset';,
    TITLE=Type,
    DESCRIPTION='Type of Financial Line Item', $
  DEFINE MOTEXT/MT=MONTH;,$

```

CENTFIN Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS      FILE CENTFIN  ON 05/15/03 AT 10.25.52

          ROOT_SEG
01          S4
*****
*YEAR          **
*QUARTER       **
*MONTH         **
*ITEM          **
*              **
*****
*****

```

CENTHR Master File

```

FILE=CENTHR, SUFFIX=FOC
SEGNAME=EMPSEG, SEGTYPE=S1, $
FIELD=ID_NUM, ALIAS=ID#, FORMAT=I9,
  TITLE='Employee, ID#',
  DESCRIPTION='Employee Identification Number', $
FIELD=LNAME, ALIAS=LN, FORMAT=A14,
  TITLE='Last, Name',
  DESCRIPTION='Employee Last Name', $
FIELD=FNAME, ALIAS=FN, FORMAT=A12,
  TITLE='First, Name',
  DESCRIPTION='Employee First Name', $
FIELD=PLANT, ALIAS=PLT, FORMAT=A3,
  TITLE='Plant, Location',
  DESCRIPTION='Location of the manufacturing plant',
  WITHIN='*Location', $
FIELD=START_DATE, ALIAS=SDATE, FORMAT=YYMD,
  TITLE='Starting, Date',
  DESCRIPTION='Date of employment', $
FIELD=TERM_DATE, ALIAS=TERM_DATE, FORMAT=YYMD,
  TITLE='Termination, Date',
  DESCRIPTION='Termination Date', $
FIELD=STATUS, ALIAS=STATUS, FORMAT=A10,
  TITLE='Current, Status',
  DESCRIPTION='Job Status', $
FIELD=POSITION, ALIAS=JOB, FORMAT=A2,
  TITLE=Position,
  DESCRIPTION='Job Position', $
FIELD=PAYSCALE, ALIAS=PAYLEVEL, FORMAT=I2,
  TITLE='Pay, Level',
  DESCRIPTION='Pay Level',
  WITHIN='*Wages', $
DEFINE POSITION_DESC/A17=IF POSITION EQ 'BM' THEN
  'Plant Manager' ELSE
  IF POSITION EQ 'MR' THEN 'Line Worker' ELSE
  IF POSITION EQ 'TM' THEN 'Line Manager' ELSE
  'Technician';
  TITLE='Position, Description',
  DESCRIPTION='Position Description',
  WITHIN='PLANT', $
DEFINE BYEAR/YY=START_DATE;
  TITLE='Beginning, Year',
  DESCRIPTION='Beginning Year',
  WITHIN='*Starting Time Period', $

```

```

DEFINE BQUARTER/Q=START_DATE;
  TITLE='Beginning,Quarter',
  DESCRIPTION='Beginning Quarter',
  WITHIN='BYEAR',
DEFINE BMONTH/M=START_DATE;
  TITLE='Beginning,Month',
  DESCRIPTION='Beginning Month',
  WITHIN='BQUARTER', $
DEFINE EYEAR/YY=TERM_DATE;
  TITLE='Ending,Year',
  DESCRIPTION='Ending Year',
  WITHIN='*Termination Time Period', $
DEFINE EQUARTER/Q=TERM_DATE;
  TITLE='Ending,Quarter',
  DESCRIPTION='Ending Quarter',
  WITHIN='EYEAR', $
DEFINE EMONTH/M=TERM_DATE;
  TITLE='Ending,Month',
  DESCRIPTION='Ending Month',
  WITHIN='EQUARTER', $
DEFINE RESIGN_COUNT/I3=IF STATUS EQ 'RESIGNED' THEN 1
  ELSE 0;
  TITLE='Resigned,Count',
  DESCRIPTION='Resigned Count', $
DEFINE FIRE_COUNT/I3=IF STATUS EQ 'TERMINAT' THEN 1
  ELSE 0;
  TITLE='Terminated,Count',
  DESCRIPTION='Terminated Count', $
DEFINE DECLINE_COUNT/I3=IF STATUS EQ 'DECLINED' THEN 1
  ELSE 0;
  TITLE='Declined,Count',
  DESCRIPTION='Declined Count', $
DEFINE EMP_COUNT/I3=IF STATUS EQ 'EMPLOYED' THEN 1
  ELSE 0;
  TITLE='Employed,Count',
  DESCRIPTION='Employed Count', $
DEFINE PEND_COUNT/I3=IF STATUS EQ 'PENDING' THEN 1
  ELSE 0;
  TITLE='Pending,Count',
  DESCRIPTION='Pending Count', $
DEFINE REJECT_COUNT/I3=IF STATUS EQ 'REJECTED' THEN 1
  ELSE 0;
  TITLE='Rejected,Count',
  DESCRIPTION='Rejected Count', $
DEFINE FULLNAME/A28=LNAME||', '||FNAME;
  TITLE='Full Name',
  DESCRIPTION='Full Name: Last, First', WITHIN='POSITION_DESC', $

```

```
DEFINE SALARY/D12.2=IF BMONTH LT 4 THEN PAYLEVEL * 12321
ELSE IF BMONTH GE 4 AND BMONTH LT 8 THEN PAYLEVEL * 13827
ELSE PAYLEVEL * 14400;,
TITLE='Salary',
DESCRIPTION='Salary', $
DEFINE PLANTLNG/All=DECODE PLANT (BOS 'Boston' DAL 'Dallas'
LA 'Los Angeles' ORL 'Orlando' SEA 'Seattle' STL 'St Louis'
ELSE 'n/a');$
```

CENTHR Structure Diagram

SECTION 01

STRUCTURE OF FOCUS FILE CENTHR ON 05/15/03 AT 10.40.34

```
EMPSEG
01      S1
*****
*ID_NUM      **
*LNAME       **
*FNAME       **
*PLANT       **
*            **
*****
*****
```

CENTINV Master File

```

FILE=CENTINV, SUFFIX=FOC, FDFC=19, FYRT=00
SEGNAME=INVINFO, SEGTYPE=S1, $
FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
  TITLE='Product,Number:',
  DESCRIPTION='Product Number', $
FIELD=PRODNAME, ALIAS=PNAME, FORMAT=A30,
  WITHIN=PRODCAT,
  TITLE='Product,Name:',
  DESCRIPTION='Product Name', $
FIELD=QTY_IN_STOCK, ALIAS=QIS, FORMAT=I7,
  TITLE='Quantity,In Stock:',
  DESCRIPTION='Quantity In Stock', $
FIELD=PRICE, ALIAS=RETAIL, FORMAT=D10.2,
  TITLE='Price:',
  DESCRIPTION=Price, $
FIELD=COST, ALIAS=OUR_COST, FORMAT=D10.2,
  TITLE='Our,Cost:',
  DESCRIPTION='Our Cost:', $
DEFINE PRODCAT/A22 = IF PRODNAME CONTAINS 'LCD'
  THEN 'VCRs' ELSE IF PRODNAME
  CONTAINS 'DVD' THEN 'DVD' ELSE IF PRODNAME CONTAINS 'Camcor'
  THEN 'Camcorders'
  ELSE IF PRODNAME CONTAINS 'Camera' THEN 'Cameras' ELSE IF PRODNAME
  CONTAINS 'CD' THEN 'CD Players'
  ELSE IF PRODNAME CONTAINS 'Tape' THEN 'Digital Tape Recorders'
  ELSE IF PRODNAME CONTAINS 'Combo' THEN 'Combo Players'
  ELSE 'PDA Devices'; WITHIN=PRODTYPE, TITLE='Product Category:', $
DEFINE PRODTYPE/A19 = IF PRODNAME CONTAINS 'Digital' OR 'DVD' OR 'QX'
  THEN 'Digital' ELSE 'Analog'; WITHIN='*Product Dimension',
  TITLE='Product Type:', $

```

CENTINV Structure Diagram

```

SECTION 01
STRUCTURE OF FOCUS FILE CENTINV ON 05/15/03 AT 10.43.35

      INVINFO
01      S1
*****
*PROD_NUM      **I
*PRODNAME      **
*QTY_IN_STOCK**
*PRICE         **
*              **
*****
*****

```

CENTORD Master File

```

FILE=CENTORD, SUFFIX=FOC
SEGNAME=OINFO, SEGTYPE=S1, $
FIELD=ORDER_NUM, ALIAS=ONUM, FORMAT=A5, INDEX=I,
  TITLE='Order,Number:',
  DESCRIPTION='Order Number', $
FIELD=ORDER_DATE, ALIAS=ODATE, FORMAT=YYMD,
  TITLE='Date,Of Order:',
  DESCRIPTION='Date Of Order', $
FIELD=STORE_CODE, ALIAS=SNUM, FORMAT=A6, INDEX=I,
  TITLE='Company ID#:',
  DESCRIPTION='Company ID#', $
FIELD=PLANT, ALIAS=PLNT, FORMAT=A3, INDEX=I,
  TITLE='Manufacturing,Plant',
  DESCRIPTION='Location Of Manufacturing Plant',
  WITHIN='*Location', $
DEFINE YEAR/YY=ORDER_DATE;,
  WITHIN='*Time Period', $
DEFINE QUARTER/Q=ORDER_DATE;,
  WITHIN='YEAR', $
DEFINE MONTH/M=ORDER_DATE;,
  WITHIN='QUARTER', $
SEGNAME=PINFO, SEGTYPE=S1, PARENT=OINFO, $
FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
  TITLE='Product,Number#:',
  DESCRIPTION='Product Number#', $
FIELD=QUANTITY, ALIAS=QTY, FORMAT=I8C,
  TITLE='Quantity:',
  DESCRIPTION=Quantity, $
FIELD=LINEPRICE, ALIAS=LINETOTAL, FORMAT=D12.2MC,
  TITLE='Line,Total',
  DESCRIPTION='Line Total', $
DEFINE LINE_COGS/D12.2=QUANTITY*COST;,
  TITLE='Line,Cost Of,Goods Sold',
  DESCRIPTION='Line cost of goods sold', $
DEFINE PLANTLNG/All=DECODE PLANT (BOS 'Boston' DAL 'Dallas'
  LA 'Los Angeles' ORL 'Orlando' SEA 'Seattle' STL 'St Louis'
  ELSE 'n/a');
SEGNAME=INVSEG, SEGTYPE=DKU, PARENT=PINFO, CRFILE=CENTINV,
  CRKEY=PROD_NUM, CRSEG=INVINFO, $
SEGNAME=STOSEG, SEGTYPE=DKU, PARENT=OINFO, CRFILE=CENTCOMP,
  CRKEY=STORE_CODE, CRSEG=COMPINFO, $

```

CENTORD Structure Diagram

SECTION 01

STRUCTURE OF FOCUS

FILE CENTORD ON 05/15/03 AT 10.17.52

```

OINFO
01      S1
*****
*ORDER_NUM    **I
*STORE_CODE   **I
*PLANT        **I
*ORDER_DATE   **
*
*****
      I
      +-----+
      I              I
      I STOSEG      I PINFO
02      I KU          03      I S1
.....
:STORE_CODE   :K    *PROD_NUM    **I
:STORENAME    :      *QUANTITY   **
:STATE        :      *LINEPRICE  **
:              :      *           **
:              :      *           **
:.....:      *****
JOINED  CENTCOMP *****
      I
      I
      I
      I INVSEG
      04      I KU
.....
:PROD_NUM     :K
:PRODNAME     :
:QTY_IN_STOCK:
:PRICE        :
:              :
:.....:
      JOINED  CENTINV
    
```

CENTQA Master File

```

FILE=CENTQA, SUFFIX=FOC, FDFC=19, FYRT=00
SEGNAME=PROD_SEG, SEGTYPE=S1, $
FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
  TITLE='Product,Number',
  DESCRIPTION='Product Number', $
SEGNAME=PROB_SEG, PARENT=PROD_SEG, SEGTYPE=S1, $
FIELD=PROBNUM, ALIAS=PROBNO, FORMAT=I5,
  TITLE='Problem,Number',
  DESCRIPTION='Problem Number',
  WITHIN=PLANT,$
FIELD=PLANT, ALIAS=PLT, FORMAT=A3, INDEX=I,
  TITLE=Plant,
  DESCRIPTION=Plant,
  WITHIN=PROBLEM_LOCATION,$
FIELD=PROBLEM_DATE, ALIAS=PDATE, FORMAT=YYMD,
  TITLE='Date,Problem,Reported',
  DESCRIPTION='Date Problem Was Reported', $
FIELD=PROBLEM_CATEGORY, ALIAS=PROBCAT, FORMAT=A20, $
  TITLE='Problem,Category',
  DESCRIPTION='Problem Category',
  WITHIN=*Problem,$
FIELD=PROBLEM_LOCATION, ALIAS=PROBLOC, FORMAT=A10,
  TITLE='Location,Problem,Occurred',
  DESCRIPTION='Location Where Problem Occurred',
  WITHIN=PROBLEM_CATEGORY,$
DEFINE PROB_YEAR/YY=PROBLEM_DATE;
  TITLE='Year,Problem,Occurred',
  DESCRIPTION='Year Problem Occurred',
  WITHIN=*Time Period,$
DEFINE PROB_QUARTER/Q=PROBLEM_DATE;
  TITLE='Quarter,Problem,Occurred',
  DESCRIPTION='Quarter Problem Occurred',
  WITHIN=PROB_YEAR,$
DEFINE PROB_MONTH/M=PROBLEM_DATE;
  TITLE='Month,Problem,Occurred',
  DESCRIPTION='Month Problem Occurred',
  WITHIN=PROB_QUARTER,$
DEFINE PROBLEM_OCCUR/I5 WITH PROBNUM=1;
  TITLE='Problem,Occurrence'
  DESCRIPTION='# of times a problem occurs',$
DEFINE PLANTLNG/All=DECODE PLANT (BOS 'Boston' DAL 'Dallas'
  LA 'Los Angeles' ORL 'Orlando' SEA 'Seattle' STL 'St Louis'
  ELSE 'n/a');$
SEGNAME=INVSEG, SEGTYPE=DKU, PARENT=PROD_SEG, CRFILE=CENTINV,
  CRKEY=PROD_NUM, CRSEG=INVINFO,$

```

CENTQA Structure Diagram

SECTION 01

STRUCTURE OF FOCUS FILE CENTQA ON 05/15/03 AT 10.46.43

```

          PROD_SEG
01      S1
*****
*PROD_NUM  **I
*          **
*          **
*          **
*          **
*****
*****
          I
          +-----+
          I          I
          I INVSEG    I PROB_SEG
02      I KU          03      I S1
.....
:PROD_NUM   :K  *PROBNUM    **
:PRODNAME   :   *PLANT      **I
:QTY_IN_STOCK: *PROBLEM_DATE**
:PRICE      :   *PROBLEM_CAT>**
:           :   *          **
:.....:   *****
JOINED CENTINV *****

```

CENTGL Master File

```

FILE=CENTGL ,SUFFIX=FOC
SEGNAME=ACCOUNTS, SEGTYPE=S1
FIELDNAME=GL_ACCOUNT, ALIAS=GLACCT, FORMAT=A7,
TITLE='Ledger,Account', FIELDTYPE=I, $
FIELDNAME=GL_ACCOUNT_PARENT, ALIAS=GLPAR, FORMAT=A7,
TITLE=Parent,
PROPERTY=PARENT_OF, REFERENCE=GL_ACCOUNT, $
FIELDNAME=GL_ACCOUNT_TYPE, ALIAS=GLTYPE, FORMAT=A1,
TITLE=Type,$
FIELDNAME=GL_ROLLUP_OP, ALIAS=GLROLL, FORMAT=A1,
TITLE=Op, $
FIELDNAME=GL_ACCOUNT_LEVEL, ALIAS=GLLEVEL, FORMAT=I3,
TITLE=Lev, $
FIELDNAME=GL_ACCOUNT_CAPTION, ALIAS=GLCAP, FORMAT=A30,
TITLE=Caption,
PROPERTY=CAPTION, REFERENCE=GL_ACCOUNT, $
FIELDNAME=SYS_ACCOUNT, ALIAS=ALINE, FORMAT=A6,
TITLE='System,Account,Line', MISSING=ON, $

```

CENTGL Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE CENTGL   ON 05/15/03 AT 15.18.48

          ACCOUNTS
01      S1
*****
*GL_ACCOUNT  **I
*GL_ACCOUNT_>**
*GL_ACCOUNT_>**
*GL_ROLLUP_OP**
*
*****
*****
```

CENTSYSF Master File

```
FILE=CENTSYSF ,SUFFIX=FOC
SEGNAME=RAWDATA ,SEGTYPE=S2
  FIELDNAME = SYS_ACCOUNT , ,A6 , FIELDTYPE=I,
    TITLE='System,Account,Line', $
  FIELDNAME = PERIOD , ,YYM , FIELDTYPE=I,$
  FIELDNAME = NAT_AMOUNT , ,D10.0 , TITLE='Month,Actual', $
  FIELDNAME = NAT_BUDGET , ,D10.0 , TITLE='Month,Budget', $
  FIELDNAME = NAT_YTDAMT , ,D12.0 , TITLE='YTD,Actual', $
  FIELDNAME = NAT_YTDBUD , ,D12.0 , TITLE='YTD,Budget', $
```

CENTSYSF Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE CENTSYSF   ON 05/15/03 AT 15.19.27

          RAWDATA
01      S2
*****
*SYS_ACCOUNT **I
*PERIOD      **I
*NAT_AMOUNT  **
*NAT_BUDGET  **
*
*****
*****
```

CENTSTMT Master File

```

FILE=CENTSTMT, SUFFIX=FOC
SEGNAME=ACCOUNTS, SEGTYPE=S1
  FIELD=GL_ACCOUNT, ALIAS=GLACCT,  FORMAT=A7,
    TITLE='Ledger,Account', FIELDTYPE=I, $
  FIELD=GL_ACCOUNT_PARENT, ALIAS=GLPAR, FORMAT=A7,
    TITLE=Parent,
    PROPERTY=PARENT_OF, REFERENCE=GL_ACCOUNT, $
  FIELD=GL_ACCOUNT_TYPE, ALIAS=GLTYPE, FORMAT=A1,
    TITLE=Type,$
  FIELD=GL_ROLLUP_OP, ALIAS=GLROLL, FORMAT=A1,
    TITLE=Op, $
  FIELD=GL_ACCOUNT_LEVEL, ALIAS=GLLEVEL, FORMAT=I3,
    TITLE=Lev, $
  FIELD=GL_ACCOUNT_CAPTION, ALIAS=GLCAP, FORMAT=A30,
    TITLE=Caption,
    PROPERTY=CAPTION, REFERENCE=GL_ACCOUNT, $
SEGNAME=CONSOL, SEGTYPE=S1, PARENT=ACCOUNTS, $
  FIELD=PERIOD, ALIAS=MONTH, FORMAT=YM, $
  FIELD=ACTUAL_AMT, ALIAS=AA, FORMAT=D10.0, MISSING=ON,
    TITLE='Actual', $
  FIELD=BUDGET_AMT, ALIAS=BA, FORMAT=D10.0, MISSING=ON,
    TITLE='Budget', $
  FIELD=ACTUAL_YTD, ALIAS=AYTD, FORMAT=D12.0, MISSING=ON,
    TITLE='YTD,Actual', $
  FIELD=BUDGET_YTD, ALIAS=BYTD, FORMAT=D12.0, MISSING=ON,
    TITLE='YTD,Budget', $

```

CENTSTMT Structure Diagram

SECTION 01

STRUCTURE OF FOCUS

FILE CENTSTMT ON 05/15/03 AT 14.45.44

```
          ACCOUNTS
01      S1
*****
*GL_ACCOUNT  **I
*GL_ACCOUNT_>**
*GL_ACCOUNT_>**
*GL_ROLLUP_OP**
*
*
*****
          I
          I
          I
          I CONSOL
02      I S1
*****
*PERIOD      **
*ACTUAL_AMT  **
*BUDGET_AMT  **
*ACTUAL_YTD  **
*
*
*****
          *****
```

B | Error Messages

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.

- ❑ For CMS, the file type is ERRORS.
- ❑ For z/OS, the ddname is ERRORS.

Topics:

- ❑ Accessing Error Files
- ❑ Displaying Messages

Accessing Error Files

For CMS, the ERRORS files are:

- ❑ FOT004 ERRORS
- ❑ FOG004 ERRORS
- ❑ FOM004 ERRORS
- ❑ FOS004 ERRORS
- ❑ FOA004 ERRORS
- ❑ FSQLXLT ERRORS
- ❑ FOCSTY ERRORS
- ❑ FOB004 ERRORS

For z/OS, these files are the following members in the ERRORS PDS:

- ❑ FOT004
- ❑ FOG004
- ❑ FOM004
- ❑ FOS004
- ❑ FOA004
- ❑ FSQLXLT
- ❑ FOCSTY
- ❑ FOB004

Displaying Messages

To display the text and explanation for any message, issue the following query command at the FOCUS command level

```
? n
```

where:

n

Is the message number.

The message number and text appear, 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.

C Table Syntax Summary

This appendix summarizes FOCUS reporting commands and options.

Topics:

- ❑ TABLE Syntax Summary
- ❑ TABLEF Syntax Summary
- ❑ MATCH Syntax Summary
- ❑ FOR Syntax Summary

TABLE Syntax Summary

The syntax of a TABLE request is:

```

DEFINE FILE filename      CLEAR|ADD
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] sortfieldn [IN-GROUPS-OF qty]
  [NOPRINT| AS 'title1,...,title5']
BY [HIGHEST] sortfieldn [IN-GROUPS-OF qty]
  [NOPRINT| AS 'title1,...,title5']
BY [HIGHEST|LOWEST{n}] TOTAL [prefix_operator] {field|code_value}
RANKED [AS 'name'] BY {TOP|HIGHEST|LOWEST} [n] field
  [PLUS OTHERS AS 'othertext']
  [IN-GROUPS-OF qty [TILES [TOP m]] [AS 'heading']]
  [NOPRINT|AS 'title1,...,title5']

{BY|ACROSS} sortfield IN-RANGES-OF value [TOP limit]
ON sfld option1 [AND] option2 [WHEN expression;...]
ON sfld RECAP fld1 [/fmt] = FORECAST (fld2, intvl, npredct,
  '{MOVAVE|EXPAVE}', npnt);
ON sfld RECAP fld1[/fmt] = FORECAST(fld2, interval, npredict, 'DOUBLEXP',
  npoint1, npoint2);
ON sfld RECAP fld1[/fmt] = FORECAST(fld2, interval, npredict, 'SEASONAL',
  nperiod, npoint1, npoint2, npoint3);
ON sfld RECAP fld1 [/fmt] = FORECAST (fld2, intvl, npredct,
  'REGRESS');

```

```

ON {sortfield|TABLE} RECAP y[/fmt] = REGRESS(n, x1, [x2, [x3,]] z);
ON sfld RECAP fld1 [/fmt] = FORECAST (infield, interval, npredict,
'DOUBLEXP', npoint, npoint2);
ON sfld RECAP fld1 [/fmt] = FORECAST (infield, interval, npredict,
'SEASONAL', nperiod, npoint, npoint2, npoint3); {BY|ON} fieldname
SUBHEAD
  [NEWPAGE]
"text"

{BY|ON} fieldname SUBFOOT [WITHIN] [MULTILINES][NEWPAGE]
"text" [<prefop.fieldname ... >] [WHEN expression;]

WHERE [TOTAL] expression
WHERE {RECORDLIMIT|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|SAVE|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|ACROSS-TOTAL}[/R|/L|/C][format] [AS 'name'] fldname
{BY|ON} sfld [AS 'text1'] {SUBTOTAL|SUB-TOTAL|SUMMARIZE|RECOMPUTE}
[MULTILINES] [pref. ] [field1 [pref. ] field2 ...] [AS 'text2']
[WHEN expression;]
{ACROSS|ON} sfld [AS 'text1'] {SUBTOTAL|SUB-TOTAL|SUMMARIZE|RECOMPUTE}
[AS 'text2'] [COLUMNS c1 [AND c2 ...]]
ON TABLE {SUBTOTAL|SUB-TOTAL|SUMMARIZE|RECOMPUTE}
[pref. ] [field1 [pref. ] field2 ...] [AS 'text2']
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 sortfield1]
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"
parentvalue {GET|WITH} CHILD[REN] [{n|ALL}] [ADD [m|ALL]]
  [AS {CAPTION|'text'}] [LABEL label]parentvalue ADD [{m|ALL}] [AS
{CAPTION|'text'}] [LABEL label]
PAGE-BREAK [OVER]
```

tag

Can be any of the following:

```
value [OR value...] value TO value
```

options

Can be any of the following:

```
AS 'text'
[INDENT m]
NOPRINT
[LABEL label]
WHEN EXISTS
[POST [TO filename]]
```

D Writing User-Coded Programs to 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 to the user program one at a time.

Topics:

- ▣ Arguments Used in Calls to Programs That Create HOLD Files

Arguments Used in Calls to Programs That Create HOLD Files

Call the program 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 been processed normally. If RETCOD is non-zero, FOCUS terminates the report and display:

`(FOC350) ERROR WRITING OUTPUT FILE:`

The error message includes the non-zero value of RETCOD.

- ❑ ACVT is a one-word integer; reserved.

In z/OS, the subroutine must be allocated to the 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 should be 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 executes 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.

```


Index

- _ masking character 180
- SET command 324
- ? DEFINE command 216
- ? FILTER command 199, 200
- ? JOIN command 873
- ? STAT command 147
- ? STYLE command 517
- ? STYLE query command 517
- ?F command 43
- ?FF command 44
- [F]DEFCENT attribute 339
- [F]YRTHRESH attribute 339
- * multiplication operator 327, 329
- ** exponentiation operator 329
- subtraction operator 329
- / division operator 327, 329
- % masking character 180, 182, 183
- + addition operator 329
- \$ masking character 180, 182
- \$* masking character 180, 182, 184

- OX spot marker 376

- A**
- absolute starting positions 712

- ACCEPT attribute 441, 442
- accessing help 35
- ACROSS attribute 541
- ACROSS COLUMNS AND phrase 119, 121
- ACROSS field 541
- ACROSS phrase 47, 64, 97, 107, 108, 110, 228, 1033, 1034
 - GRAPH 1033, 1034
- ACROSS summary commands 313
- ACROSS values 276, 277, 549
- ACROSS with ROW-TOTAL 274
- ACROSS-TOTAL component 276, 277
- ACROSSCOLUMN attribute 529, 532, 599, 600, 601
 - using with WHEN 599, 600, 601
- ACROSSLINE parameter 108
- ACROSSPRT parameter 111
- ACROSSTITLE component 550, 551
- ACROSSVALUE component 541, 550, 551
- ADD command 53
- ADD option 215
- ADD parameter 982
- adding blank rows 968
- adding calculated values in financial reports 971
- adding columns to financial reports 971
- adding data to financial reports 941, 952
- adding financial data to reports 952

- adding graphics to reports 773, 774, 780
- adding images to reports 773, 774, 780
- adding rows to financial reports 941, 944
- adding tag rows to financial reports 941, 942, 978
- adding text rows to financial reports 968
- adding Type 1 PostScript fonts 692
- adding underlines to financial reports 993
- adding values 54
- adding values for numeric fields 54
- adding virtual fields 216
- addition operator 329
- addressing columns 961
- aggregate values 167
- aggregation 153, 418
- aggregation and external sorting 153, 154, 155
- aliases 40, 41, 44, 1098
 - displaying 44
- aliases and SQL Translator 1098
- aligning decimal points 758, 760, 762
- aligning headings and footings 742, 744, 745, 746, 747
 - in HTML reports 744
- aligning text fields 753
- alignment methods 742
- ALL parameter 166, 826, 836
- ALL parameter and JOIN command 835, 836
- ALL parameter and missing values 826, 827
- ALL parameters 826
- ALL prefix 825
- ALL prefix and missing values 825
- ALLOCATE command 422
- ALPHA format 456
- alphanumeric fields 179, 180, 181, 182, 183, 184, 185, 186, 287
- alphanumeric fields and text fields 263
- ALT attribute 774
- alternate file views 904, 905
- alternate indexes 204
- AND operator 169, 352
- ANSI-compliant reports with no records 419
- AnV fields 347
- applying CSS styles 614
- applying grids to a report 772
- arithmetic expressions 327
- arithmetic operators 327, 329
- AS CAPTION phrase 978, 979
- AS phrase 95, 406, 407, 423, 436
 - in extract files 95
- ascending sort order 116, 117
- ASNAMES command 434, 435, 436, 437, 438
- ASQ calculations on field values 63
- ASQ prefix operators 63
- assigning row labels 955
- assigning row titles 990, 991
- assigning row titles PICKUP rows 1005
- assigning row titles RECAP rows 992
- assigning row titles TAG rows 991
- assigning titles to rows 990, 991, 992
- attribute inheritance 493, 581, 582, 584
 - augmenting attributes 582
 - overriding 584
- AUTOINDEX parameter 907, 908, 909

- AUTOPATH parameter 907
- AUTOTABLEF parameter 67, 68
- AUTOTICK parameter 1060, 1062, 1063, 1081
- AVE field 63
- AVE prefix operator 63
- AVE prefix operators 63
- B**
- BACK command in Hot Screen 86
- background images 773, 780
- BACKIMAGE attribute 773, 780
- bar charts 1048
- BAR command 993
- bar rows 993
- BARNUMB parameter 1050, 1066, 1081
- BARSPACE parameter 1048, 1050, 1081
- BARWIDTH parameter 1048, 1050, 1081
- base dates 332, 333
- BASEURL SET parameter 794
- BINARY format 457
- BINARY HOLD format 457
- BINARY output file format 457
- BINARY PCHOLD format 457
- BINS parameter 147
- blank lines 401, 402, 403, 565, 574
 - formatting 574
 - inserting 402, 403
- blank rows 968
- blank spaces 706, 716
 - above data values 716
- blanks 817
- Boolean expressions 351, 353
- Boolean operator 352
- Boolean operators 351
- BORDER attribute 768, 770
- borders 767, 768, 770
 - around headings and footings 768
 - colors 768
 - styles 768
 - width of 768
- BOTTOM command in Hot Screen 85
- BOTTOMGAP attribute 707, 715
- BOTTOMMARGIN attribute 515, 516
- boundaries for Pooled Tables 919
- boundaries sub pool 917
- browser fonts 524
- browser support for Cascading Style Sheets 607, 625, 626
- browser titles 549
- BSTACK parameter 1048, 1081
- BY 103
- BY attribute 534
- BY field in Hot Screen 90
- BY phrase 47, 55, 64, 97, 99, 100, 102, 113, 950, 1033, 1034
 - GRAPH 1033, 1034
- BY phrase using with financial reports 950, 951
- BY ROWS OVER phrase 119, 120
- BY TOTAL phrase 135, 136, 137
- BYDISPLAY parameter 102
- BYLASTPAGE system variable 554
- BYPANEL parameter 90

BYSCROLL parameter 87

byte precision 57, 58

C

CA TELLAGRAF Interface 1074

CACHEFIELDS phrase 661, 662

calculated values 206, 208, 228, 229, 270, 273, 283, 284, 285, 286, 971

calculated values and row totals 270

calculated values in column totals 273

calculated values in row totals 273

calculating column and row totals 270

calculating column percents 64

calculating column totals 270, 271, 272, 273

calculating dates 334

calculating MAX field values 64

calculating maximum field values 64

calculating maximum values for field values 64

calculating MIN field values 64

calculating minimum field values 64

calculating minimum values for field values 64

calculating row percents 64

calculating row totals 270, 271, 272

calculating trends with FORECAST 239

calculating values for temporary fields 224

calculation on field values 66

calculations 966

calculations and functions 966, 967

calculations counting field values 71

calculations on counting field values 71

calculations on field values 60, 63, 64, 113

calculations on sum numeric field values 71

calculations on SUM numeric field values 71

calculations on TOT field values 71

calculations on total field values 71

calling functions 966, 967

CAPTION parameter 977, 982, 988, 989

captions 549

captions in a hierarchy 981

captions in Master Files 977, 988, 989

CAR data source 1126, 1127

Cartesian product 900, 901, 902

Cartesian product answer sets 1102

Cartesian product answer sets and SQL Translator 1102

Cascading Style Sheet classes 609, 610, 614, 615, 617, 618, 625, 627
assigning to report components 614, 615, 627
naming 610, 615

Cascading Style Sheet rules 609, 618, 625

Cascading Style Sheets (CSS) 492, 495, 496, 605, 606, 607, 609, 610, 612, 613, 614, 615, 616, 617, 618, 620, 621, 622, 623, 624, 625, 626, 627

browser support 607, 625, 626

conditional styling 615, 620, 621, 625

external 496, 607, 609, 614

formatting 614, 620

hyperlinks 621

images 621

inheritance 610, 627

internal 607, 612, 620

linking to 617, 618, 622, 623, 624

naming classes 610, 615

- Cascading Style Sheets (CSS) (*continued*)
 - report formatting 616, 617
 - rules 609, 618, 625
 - troubleshooting 626
- CDN (Continental Decimal Notation) 1102
- CDN (Continental Decimal Notation) and SQL Translator 1103
- cell formatting 992
- cell notation 965
- cells 965
- CENTCOMP data source 1160
- CENTFIN data source 1158
- CENTGL data source 1169, 1170
- CENTHR data source 1158
- CENTINV data source 1158
- CENTORD data source 1158
- CENTQA data source 1158
- CENTSTMT data source 1171, 1172
- CENTSYSF data source 1170
- Century Corp data sources 1158, 1160, 1161, 1162, 1164, 1165, 1166, 1167, 1168, 1169
- changing row titles 990, 991
- changing row titles PICKUP rows 1005
- changing row titles RECAP rows 992
- changing row titles TAG rows 991
- character expressions 325, 344
- character strings 88, 179, 345, 346
 - locating 88
- chart of accounts hierarchies 975, 977
- charts of accounts 978
- CHECK FILE command 872
- CHECK FILE command and join structures 871, 872
- CHECK PICTURE command 51
- CHECK STYLE command 508
- CLASS attribute 614, 615, 617, 618, 625
- class intervals 1062
- classes in Cascading Style Sheets 609, 614, 615, 617, 618, 625, 627
 - assigning to report components 609, 614, 615, 618, 625, 627
- clearing conditional join structures 874, 875
- clearing join structures 874, 875
- clearing virtual fields 216, 217
- clusters 917
- CMS requirements 147
- CNOTATION SET parameter 229, 230, 962, 963
- CNT prefix operator 71
- collapsing PRINT with ACROSS 111
- COLOR attribute 519
- color values 522
- COLSPAN attribute 748
- column addresses 961, 962
- column and row totals in calculated values 270
- COLUMN attribute 529, 530, 540
- column notation 229, 230, 962, 963
- column numbers 959
- column reference numbers 229, 962
- column spacing 405
- column titles 406, 407, 408, 409, 410, 549, 551
 - creating 406, 407
 - customizing 408
 - identifying 551
 - justifying 409, 410

- column totals 270, 272, 273
- column values 964
- column width 644, 646
- COLUMN-TOTAL phrase 270, 271, 272
- columns 76, 395, 396, 397, 398, 399, 400, 405, 529, 530, 532, 540, 549, 550, 551, 644, 706, 707, 717, 719, 720, 721, 732
 - and SQUEEZE attribute 644
 - compressing 395, 396, 397
 - controlling order 717
 - controlling spacing 717
 - determining width 717, 719, 720, 721
 - formatting 76
 - identifying in a style sheet 530, 532, 540
 - justifying 732
 - positioning 398, 399, 400, 706, 707
 - spacing 405
 - stacking 717
 - titles of 549, 550, 551
 - width 644
- columns in financial reports 941
- columns numbers 959
- COLUMNS parameter 93
- COM format 458
- COM HOLD format 458
- COM output file format 458
- COM PCHOLD format 458
- COM SAVE format 458
- COMASTER Master File 1144
- combination of summary commands 307
- combinations of subtotals 306
- combining CSS with other formatting methods 620, 625
- combining expressions 170
- combining fields in date expressions 336, 337
- combining multiple values 944
- combining range of records 946
- combining range of values 946, 947
- combining records 944, 945
- combining values 944, 945, 947
- COMMA format 457
- COMMA HOLD format 457
- COMMA output file format 457
- COMMA SAVE format 457
- comma-delimited files 834
- commands 88, 367, 381, 387, 388, 389, 391, 395, 396, 401, 403, 599
 - canceling in Hot Screen 88
 - FOLDLINE 395
 - MULTILINES 367
 - NOPRINT 391
 - NOSPLIT 387, 388, 389
 - OVER 396
 - PAGEBREAK 381
 - REPAGE 381
 - repeating in Hot Screen 88
 - SKIP-LINE 401
 - SUBFOOT 367
 - SUPPRINT 391
 - UNDERLINE 403
 - WHEN 599
- comments 507
- common high-order sort fields 884, 885, 887, 897, 899
- comparing characters with masks 180, 181, 182, 183, 184
- comparing records 881, 882, 883
- compiling calculated values 914

- compiling expressions 912, 913
- compiling virtual fields 912
- complex expressions 323
- COMPMISS parameter 78, 820
- compound expressions 169, 170
- compound reports 466, 689, 690
 - displaying 689
- COMPUTE command 55, 224, 225, 226, 228, 283, 302, 315, 316, 318, 324
- COMPUTE command expressions 324
- computing the average field values 63
- COMT format 459
- COMT HOLD format 459
- COMT output file format 459
- COMT PCHOLD format 459
- COMT SAVE format 459
- concatenated data sources and MATCH FILE command 895
- concatenating character strings 346
- concatenating data sources 890, 891, 892
- concatenating data sources and field names 894
- concatenation 345, 1036
 - MORE phrase 1036
 - universal 1036
- concatenation data sources 892, 895
- concatenation for data sources 890
- concatenation operators 345
- concatenation usage formats 893
- conditional drill-down 789, 790
- conditional expression types 353
- conditional expressions 325, 353, 354, 355
 - conditional formatting 411, 412, 413, 414, 415, 416, 593, 598
 - conditional grid formatting 603
 - conditional join structures 832, 834, 843, 861, 862, 863, 870
 - conditional links 789, 790
 - conditional operators 171, 175, 176
 - conditional styling 585, 586, 587, 588, 589, 590, 615, 620, 625
 - sequential conditional formatting 586
 - StyleSheets and 585, 587, 588, 589, 590
 - conditional text 320, 321
- configuring PostScript fonts in z/OS 693
- connected point plot graphs 1041, 1042
- consolidating financial data 978, 981, 982, 984, 985
 - single row 984
- consolidating financial data in multiple rows 985
- consolidating financial data single row 982
- constant dates 331, 335
- constants 953
 - constants in date expressions 335
 - constants in Financial Modeling Language (FML) 952
 - constants in financial reports 952
 - constants in FML (Financial Modeling Language) 952
 - constants in FML requests 952, 953
- CONTAINS operator 179
- contiguous columns 960
- contiguous columns in financial reports 960

- Continental Decimal Notation (CDN) 1102
- Continental Decimal Notation (CDN) and SQL Translator 1102, 1103
- controlling attributes 434, 439, 440, 441, 442
- controlling attributes and HOLD Master Files 440
- controlling column reference numbers 229, 962
- controlling field names 435, 436, 437, 438
- controlling field names HOLD Master Files 435
- controlling fields 439
- converting data types for join structures 861
- COUNT * command 56, 57
- COUNT command 55, 56, 98
- COUNT command for unique segments 56
- count of occurrences 71
- counting field values 55
- COUNTWIDTH SET parameter 55, 57, 58
- COURSE data source 1136, 1137
- COURSES data source 1132
- CREATE TABLE command 1099, 1100
- CREATE VIEW command 1100, 1101
- creating calculated values 208, 224, 225, 226
- creating financial reports 939, 944
- creating FOCUS data sources 429, 432, 433, 434
- creating free-form reports 1010, 1014, 1015, 1016
- creating HOLD files 423, 424, 425, 426, 429, 430, 432, 1183
- creating multiple virtual fields 215
- creating numeric expressions 327
- creating output files 423
- creating PCHOLD files 453
- creating RECAP expressions 953
- creating reports 31, 32, 33, 34, 36
- creating rows 941, 942, 943
- creating rows from multiple records 944
- creating rows in financial reports 941
- creating SAVB files 452
- creating SAVE files 449, 450, 451
- creating tag rows 942, 943
- creating temporary fields 36
- creating temporary fields with COMPUTE phrases 224
- creating temporary fields with DEFINE FUNCTION 264, 265, 267
- creating virtual fields 208, 209, 210, 211, 213, 215, 218
- cross-century dates 335
- cross-referenced fields 853
- cross-referenced files 835, 843
- CSS (Cascading Style Sheets) 492, 495, 496, 605, 606, 607, 609, 610, 612, 613, 614, 615, 616, 617, 618, 620, 621, 622, 623, 624, 625, 626, 627
 - browser support 607, 625, 626
 - conditional styling 615, 620, 621, 625
 - external 496, 607, 609, 614
 - formatting 614, 620
 - hyperlinks 621
 - images 621
 - inheritance 610, 627
 - internal 607, 612, 620
 - linking to 617, 618, 622, 623, 624
 - naming classes 610, 615
 - report formatting 616, 617

CSS (Cascading Style Sheets) (*continued*)
troubleshooting 626

CSSURL attribute 617, 623, 624

CSSURL parameter 617, 618, 622

custom report titles 549

custom reports 499

custom sort order 119

custom worksheet names 549

customizing reports 38, 357, 394, 410, 411, 1014
with SET parameters 410

customizing sort order 119

D

data 32, 644

- descriptions 32
- source types 32
- wrapping 644

DATA component 538, 539

data extraction 423, 426

data fields 180

data formats for cross-referenced fields 859

data formats for host fields 859

data formats for join structures 859

data retrieval 162, 164, 903, 904, 905, 907, 908,
909, 911, 941, 1004, 1005

data retrieval and TABLEF command 910, 911

data retrieval using TABLEF command 910

data source 51

data sources 32, 34, 36, 878, 890, 1113
joining 36
merging 878
types 32

data structures 906

data type conversion 860

data type conversions 860, 861

data types 264, 860, 861

data wrapping 644, 646

date constants 331

date expressions 331, 336

date fields 331

date formats 641, 643, 1104, 1105
unsupported 643

date formats and SQL Translator 1104, 1105

date value formats 332, 333

date values 332

date-time data types 337

date-time expression types 331

date-time expressions 331

date-time field formats 337

date-time format 342

date-time format and display fields 337

date-time values 331, 1106, 1107, 1108

date-time values and SQL Translator 1106, 1107,
1108

DATEFORMAT parameter 337

DATEFORMAT setting 337

dates 334, 644, 1067
in graphs 1067
separators 644

DB2 format 460

DB2 HOLD format 460

DB2 output file format 460

DBAFILE attribute 835

- decimal points 755, 758, 760, 762
 - aligning with data 755
- decimal values 758, 760, 762
 - comparing 760
- declaring filters 196
- default formatting 625
- default proportional fonts 523
- DEFAULT- FIXED attribute 524
- DEFAULT- PROPORTIONAL attribute 524
- deferred graphics output 1072
- DEFINE and dates 220, 222, 335
- DEFINE attribute 324, 1039
 - GRAPH command 1039
- DEFINE command 211, 213, 215, 324
- DEFINE command and join structures 855, 856, 857, 858, 865, 866, 868
- DEFINE command and missing values 810, 811, 812, 813, 814
- DEFINE command expressions 324
- DEFINE compiler 912, 913
- DEFINE FILE RETURN command 868
- DEFINE FILE SAVE command 219, 868
- DEFINE FUNCTION command 264, 265, 267
- DEFINE function deleting 264
- DEFINE function displaying 264, 267
- DEFINE function limitations 264, 266
- DEFINE function querying 264
- DEFINE functions 264, 265, 267
- defining custom groups 126, 127
- defining filters 195, 196, 197
- defining virtual fields 213
- DELETE command 1111
- deleting DEFINE functions 267
- delimited file, creating 473
- delimited output files 460
- descending sort order 116, 118
- designating missing values 828
- DEVICE parameter 1029, 1081
- DFIX 473
- DFIX format 460
- DFSORT utility 146, 147, 931
- DHTML HOLD format 460
- DHTML output format 460, 632
- Dialect Translation 1090
- Dialogue Manager 324, 938
- DIF format 461
- DIF output file format 461
- DIF PCHOLD format 461
- DIF SAVE format 461
- direct percent 66
- direct percent of counts (PCT.CNT) 66
- display ADD command 53
- display commands 45, 98, 888
- display commands and MATCH FILE command 888
- display COUNT * command 56
- display COUNT command 55, 98
- display field values 47, 49
- display fields 39, 60, 61, 77
 - limitations 39
- display fields for prefix operators 60
- display LIST * command 49

- display LIST command 98
- display LIST commands 45, 47
- display PRINT * command 49
- display PRINT command 98
- display PRINT commands 45, 47
- display SUM command 53, 54, 98
- display SUM commands 45
- display values 45
- display WRITE command 53
- displaying ADD command 53
- displaying all fields in a segment 50
- displaying all fields in segments 49
- displaying captions 978
- displaying children 978, 980, 981
- displaying error messages 149, 150
- displaying excluded values 103
- displaying field names 43, 44
- displaying field values 45, 47, 48, 49
- displaying FML hierarchies 975, 978
- displaying grand totals 278
- displaying graphs 1026
- displaying hierarchies 975
- displaying hierarchy values as captions 981
- displaying HOLD Master Files 423, 426
- displaying join structures 871, 872, 873
- displaying LIST * command 49
- displaying LIST command 47
- displaying LIST commands 45
- displaying parents and children 978, 980, 981
- displaying PRINT * command 49
- displaying PRINT command 47
- displaying PRINT commands 45
- displaying reports 81, 82, 89, 95, 157, 419, 496
- displaying retrieval order 51
- displaying retrieval order for multi-path data sources 51
- displaying structure for multi-path data sources 51
- displaying sub-totals 281, 282, 283
- displaying subtotals 278, 279, 280, 281, 282, 283
- displaying SUM command 53
- displaying summary lines 321
- displaying values 45
- displaying virtual fields 216
- displaying WRITE command 53
- distinct prefix operators 67
- distinguishing virtual fields and calculated values 208
- division operator 327, 329
- DOC format 461
- DOC output file format 461
- DOC PCHOLD format 461
- DOC SAVE format 461
- double exponential smoothing 251, 252
- double exponential smoothing FORECAST 251, 252
- DOWN command in Hot Screen 85
- DROP VIEW command 1100
- DROP VIEW command and SQL Translator 1101
- DST prefix operator 67

DST prefix operator restrictions 67
DST prefix operators 67
DUPLICATECOL parameter 141
dynamic reporting 975
dynamically formatting virtual fields 220, 222, 223

E

EDUCFILE data source 1121, 1122
ELEMENT attribute 801, 802
embedded fields 559, 560
embedded quotation marks 344, 345
embedding images 621
EMPDATA data source 1133, 1134, 1135
EMPLOYEE data source 1116, 1118, 1119
empty reports 419
EMPTYREPORT parameter 419
END command 35, 1032
 in GRAPH request 1032
ending a report request 35
EQ operator 178, 352
equijoins 832, 834, 845
error files 1173, 1174
 CMS 1174
 z/OS 1174
error messages 35, 1173, 1174
escape characters 184, 185, 186
ESSBASE hierarchies 975
establishing segment locations 218
estimating number of records 148
estimating records to be pooled 925
estimating report lines 925
ESTLINES 925
ESTLINES parameter 148
ESTRECORDS 925
ESTRECORDS parameter 148
evaluation of temporary fields 207
ex_forecast_dist 259
ex_forecast_mov 258
ex_forecast_mult 257
ex_regress_mult 262
Excel 2000 alignment 744
Excel 2000 format 635, 639, 657, 684
 date formatting 639
 numeric formatting 639
 PIVOT option 657
 transferring files 684
Excel 2000 TOCs (tables of contents) 671, 672
EXCEL format 461
Excel formats supported with AnV fields 461
Excel formulas 635, 655, 656, 657
 character limit 656
 generating 655
 generating calculated values 656
 PIVOT option 657
 translation support 655
excel locking 647
Excel Named Ranges 665, 666, 667
 Compound Excel reports 667
 support for 667
EXCEL output file format 461
EXCEL PCHOLD format 461
Excel reports 671, 672
EXCEL SAVE format 461
Excel workbook 665, 666, 667

- EXCEPT operator 1103
- EXCLUDES operator 187
- excluding missing values from tests 817
- existing data 178
- EXL2K alignment 744
- EXL2K display format 462
- EXL2K format 635, 637, 638, 639, 640, 641, 644, 646, 651, 653, 655, 659, 660, 662, 664, 684
 - CACHEFIELDS phrase 662
 - column width 644, 646
 - conditional styling in 637
 - data wrapping 644, 646
 - default languages for 639
 - displaying calculated values 653
 - displaying column totals 651
 - displaying dates 641
 - displaying numeric data 640
 - displaying row totals 653
 - FORMULA option 651
 - interactive spreadsheets 651
 - National Language Support 638
 - PAGEFIELDS phrase 664
 - PIVOT option 659
 - TABLE syntax 660
 - transferring files 684
 - translation support 655
- EXL2K FORMULA 462
- EXL2K FORMULA display format 462
- EXL2K output 629
- EXL2K output file format 462
- EXL2K PCHOLD format 462
- EXL2K PIVOT 463
- EXL2K PIVOT format 463
- EXL2K SAVE format 462
- EXL2K TOCs (tables of contents) 671, 672
- EXL97 display format 463
- EXL97 format 635, 685, 686, 687
 - HOLD option 685
 - limitations 687
 - StyleSheet syntax 685
 - styling reports 685, 686
- expanding byte precision 58
- expanding byte precision for COUNT command 57
- expanding precision 58
- explicit labels 955, 956, 957
- EXPN and numeric functions 328, 329
- EXPN function 328, 329
- exponential moving average 239, 241, 248, 249
- exponential moving average FORECAST 239, 241, 248, 249
- exponentiation operator 329
- exporting from data sources 423, 426
- expression dates 325, 335
- expression types 325
- expressions 206, 323, 324, 1104
- expressions and SQL Translator 1104
- expressions IF phrase 324
- expressions, relational 353
- EXTAGGR parameter 153
- extending heading and footing code 376, 377
- extending underlines 569
- external Cascading Style Sheet classes 609, 614, 615, 617, 618, 625, 626
 - assigning to report components 614, 615, 626
 - names 615
- external Cascading Style Sheet rules 609, 610, 618, 626
 - BODY element 609

external Cascading Style Sheet rules (*continued*)
 TD element 609, 610

external Cascading Style Sheets 495, 607, 609,
 610, 614, 615, 616, 617, 618, 620,
 621, 622, 623, 624, 625, 626, 627

 browser support 625, 626

 classes 609, 617, 618, 625

 combining with other formatting methods 620,
 622

 conditional styling 615, 620, 625

 hyperlinks 620, 621

 images 620, 621

 inheritance 610, 627

 linking to 617, 618, 621, 622, 623, 624

 refreshing 626

 report formatting 610, 614, 616, 617, 620,
 621, 625

 rules 609, 618, 626

 troubleshooting 626

external files 950

external sorting 146, 147, 149, 150

external sorting and aggregation 153

external sorting and HOLD files 155, 156

external sorting by aggregation 153, 154, 155

external sorting requirements 147

EXTHOLD parameter 156

extract files 422, 445, 448, 449, 481

extract files and missing values 818

EXTRACT function 1108

extracting date components 336

EXTSORT parameter 146, 147

EXTUNDERLINE attribute 569

F

field dates 331

field format expressions 326

field formats 89, 326, 335

 redefining 89

field names 40, 41, 42, 43, 44, 436, 437, 438,
 1103

 aliases 40, 41

 displaying 43, 44

 long 41, 42

 qualified 40, 41, 42

 truncated 40, 41

field names and concatenating data sources 894

field names and SQL Translator 1103

field names and universal concatenating 894

field names and universal concatenation 893

field padding 478, 479

field references for COMPUTE command 227

field reformatting and missing values 78, 820

field types not supported for EXCEL format 461

field values 45, 113, 372, 373

 embedding 372, 373

field-based reformatting 220, 222, 223

FIELDNAME command 41

fields 40, 41, 206, 326, 391, 392

 in report requests 40, 41

 suppressing display 391, 392

FILE command 34

FILEDEF command 422

FILTER parameter 195, 196, 197, 198

FILTER query command 195, 199, 200

filters 195, 196, 198, 199, 200, 201

FINANCE data source 1130

financial data 981

financial data values 941

- Financial Modeling Language (FML) 533, 552, 938, 1182
 - free text in 552
- Financial Modeling Language (FML) and Dialogue Manager 938
- financial report hierarchies 978
- financial reports 938, 975
- financial reports and external files 950
- financial reports and HOLD files 1006
- financial reports and inter-row calculations 954, 955
- financial reports and recursive models 973, 974
- financial reports charts of accounts 975, 978, 980, 981
- financial reports hierarchies 982, 988, 989, 990
- financial reports inserting text rows 968
- financial reports inserting variables in text rows 970
- financial reports records in multiple rows 949
- financial reports repeating rows 957
- financial reports saving intermediate results 1002
- financial reports sorting with BY 950
- financial reports sorting with FOR 950
- financial reports supplying data as constants 952
- fixed-axis scales (fixed limits) 1061, 1063, 1071
- FIXRETRIEVE parameter 443, 444
- FML (Financial Modeling Language) 533, 552, 938, 939, 1182
 - free text in 552
- FML (Financial Modeling Language) and Dialogue Manager and 938
- FML hierarchies 975, 977, 978, 980, 981
- FML hierarchies indenting captions 998
- FML hierarchies indenting row titles 997
- FML hierarchies indenting text or numbers 993
- FML hierarchies loading into memory 988, 989
- FOCFIELDNAME amper variable 42
- FOCPOOLT 926
- FOCPOST files 1002
- FOCSTYLE files 506, 688
- FOCUS data sources 159, 160, 429
- FOCUS file structure 429, 431
- FOCUS format 463
- FOCUS StyleSheet attributes 614, 615, 617, 618, 623, 624, 625
 - CLASS 614, 615, 617, 618, 625
 - CSSURL 617, 623, 624
- FOCUS StyleSheets 616, 620, 992
- FOLD-LINE command 718
- FOLDLINE command 395, 396, 406
- FONT attribute 523
- font files 687, 693
- font map files 687, 692, 693, 695, 696
 - entries 695
 - Portable Document Format (PDF) 695
- fonts 518, 519, 520, 521, 523, 524, 692, 693, 695, 761
 - adding PostScript fonts 692
 - adding PostScript Type 1 fonts 692
 - browser type 524
 - colors 518, 520, 521
 - configuring PostScript fonts 693
 - default type 523
 - in HTML Reports 523
 - inherited styles 520
 - measuring 761
 - PostScript (PS) 693
 - PostScript Type 1 693, 695
 - relative point sizes and HTML fonts 519

- fonts (*continued*)
 - sizes 518, 519
- footers 1037, 1039
 - embedding field values 1039
- footing code 377
- FOOTING command 1014
- FOOTING component 554, 556
- footings 358, 359, 372, 373, 378, 548, 554, 556, 559, 706, 762, 1037
 - aligning 762
 - creating 1037
 - embedded fields in 559
 - identifying in a style sheet 554, 556
 - inserting data in 372, 373, 378
 - limitations 359
 - positioning 706
- FOR phrase 67, 68, 119, 126, 127, 938, 939, 942, 950, 951, 1182
 - syntax 1182
- FOR phrase reusing values 948
- FORECAST 244, 257, 258, 259
- FORECAST double exponential smoothing 251, 252
- FORECAST exponential moving average 239, 241, 248, 249
- FORECAST limit 244
- FORECAST linear regression analysis 239, 241
- FORECAST linear regression equation 254, 256
- FORECAST processing 241
- FORECAST simple moving average 239, 241, 245, 247
- FORECAST triple exponential smoothing 252, 254
- forecasted values 598
- format ALPHA 456
- format dates 332, 333
- FORMAT DFIX 473
- format DHTML 460, 632
- formats 95
 - extracting files 95
- formatted graphs 1072
- formatting cells 992
- formatting columns 76, 77, 78, 992
- formatting data 537, 538
- formatting fields 326, 335
- formatting financial reports 992, 993
- formatting graphs 1026
- formatting heading and footing lines 757
- formatting HOLD files 423
- formatting options for financial reports 992
- formatting output files 455
- formatting PCHOLD files 453
- formatting report columns 76, 77, 78
- formatting reports 357, 359, 363, 365, 374, 380, 391, 394, 395, 396, 398, 401, 405, 408, 409, 410, 411, 494, 610, 614, 616, 617, 620, 625, 1094
- formatting reports and SQL Translator 1094
- formatting rows 992
- formatting SAVB files 452
- formatting SAVE files 449, 450
- FORMULA option 651, 653
 - displaying calculated values 653
 - displaying column totals 651
 - displaying operation results 651
 - displaying row totals 653
- FORMULTIPLE parameter 948, 949

- FORWARD command in Hot Screen 85
 - free text 548, 552, 968, 969
 - identifying in FML reports 552
 - free-form reports 614, 615, 758, 762, 1009, 1016
 - FREETEXT component 552
 - FROM ... TO operator 175, 176
 - FST prefix operator 69, 70
 - function keys in Hot Screen 92, 95
 - functions 323, 966
 - FYRTHRESH attribute
 - date-time data type and 337
- G**
- GCOLOR parameter 1081
 - GDDM graphics 1076
 - GE operator 176, 177, 352
 - generating internal Cascading Style Sheets 612, 613
 - generating TABLEF commands 1110
 - GET CHILDREN parameter 978, 982
 - GGDEMOG data source 1152
 - GGORDER data source 1152
 - GGPRODS data source 1152
 - GGSALES data source 1152
 - GGSTORES data source 1152
 - GMISSING parameter 1069, 1081
 - GMISSVAL parameter 1069, 1081
 - Gotham Grinds data sources 1152, 1153, 1154, 1155, 1156, 1157
 - GPROMPT parameter 1026, 1081
 - grand totals 278
 - GRANDTOTAL component 534, 535, 542, 543, 544
 - GRAPH command 1018, 1032, 1078
 - graph formatting 609, 610, 614, 617
 - BODY element 609
 - external Cascading Style Sheets 614, 617
 - TD element 610
 - graph forms 1040, 1041, 1042, 1045, 1048, 1053, 1056
 - bar charts 1048
 - connected point plots 1041, 1042
 - histograms 1045
 - pie charts 1053
 - scatter diagrams 1056
 - GRAPH parameters 1026, 1029, 1047, 1048, 1050, 1053, 1056, 1057, 1058, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1069, 1075, 1081
 - AUTOTICK 1060, 1081
 - BARNUMB 1050, 1066, 1081
 - BARSPACE 1048, 1050, 1081
 - BARWIDTH 1048, 1050, 1081
 - BSTACK 1048, 1081
 - DEVICE 1029, 1081
 - GCOLOR 1081
 - GMISSING 1069, 1081
 - GMISSVAL 1069, 1081
 - GPROMPT 1026, 1081
 - GRAPH 1058
 - GRIBBON 1081
 - GRID 1056, 1065, 1081
 - GTREND 1056, 1066, 1081
 - HAUTO 1061, 1081
 - HAXIS 1061, 1081
 - HCLASS 1062, 1081
 - HISTOGRAM 1047, 1081
 - HMAX 1061, 1081
 - HMIN 1061, 1081
 - HSTACK 1047, 1081
 - HTICK 1062, 1081
 - PAUSE 1075, 1081
 - PIE 1053, 1081

GRAPH parameters (*continued*)

- PLOT 1081
 - PRINT 1029, 1081
 - TERMINAL 1081
 - VAUTO 1063, 1081
 - VAXIS 1063, 1081
 - VCLASS 1064, 1081
 - VGRID 1057, 1065, 1081
 - VMAX 1063, 1081
 - VMIN 1063, 1081
 - VTICK 1064, 1081
 - VZERO 1069, 1081
- GRAPH requests 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1053, 1081
- ACROSS phrase 1033, 1034
 - BY phrase 1033, 1034
 - concatenating files 1036
 - END command 1032, 1081
 - IF phrase 1035
 - pie charts 1053
 - QUIT command 1081
- graph types 1017, 1018, 1040, 1041, 1042, 1045, 1048, 1053, 1056
- bar charts 1048
 - connected point plots 1041, 1042
 - histograms 1045
 - pie charts 1053
 - scatter diagrams 1056
- GRAPH vs. TABLE 1018, 1032
- graphic devices 1029, 1081
- graphics 773, 774, 776, 780, 781, 783, 784, 791, 792
- adding to HTML reports 783
 - adding to PDF reports 784
 - adding to reports 773, 774, 776, 781
 - linking 791, 792
- graphs 549, 1017, 1018, 1026, 1029, 1033, 1034, 1035, 1036, 1037, 1045, 1058, 1060, 1062, 1063, 1065, 1067, 1069, 1071, 1072, 1073, 1075, 1081
- adding footings 1037
 - adding grids 1065
 - adjusting parameter settings 1058, 1081

graphs (*continued*)

- annotating 1037
 - class and tick intervals 1062
 - creating from unlike data sources 1036
 - deferred output 1073
 - displaying 1026
 - displaying stored graphs 1073
 - fixed axis scales 1071
 - formatting 1026, 1081
 - headings 1037
 - horizontal axis features 1033, 1034, 1060, 1081
 - missing data 1069, 1081
 - naming subjects 1033, 1034
 - parameter settings 1081
 - plotting dates 1067
 - printer/plotter selection 1029, 1081
 - prompting for values 1026
 - redisplaying with REPLOT 1018
 - saving 1072
 - saving formatted graphs 1075
 - selecting records 1035
 - stacking bars with OVER 1045
 - titles 549
 - verb phrases 1033, 1034
 - vertical axis features 1063
- GRIBBON parameter 1081
- GRID attribute 767, 768, 771
- GRID parameter 728, 1026, 1056, 1065, 1081
- grids 767, 768, 771, 772, 1057
- group fields 854
- group fields and join structures 854
- group key values 188
- grouping numeric data 124, 125, 126, 127
- grouping numeric data into tiles 127, 129, 130, 131, 132, 133
- groups of values 947
- GT operator 176, 177, 352

GTREND parameter 1056, 1066, 1081

GUTTER attribute 801, 802

H

H data type 337

HAUTO parameter 1061, 1081

HAXIS parameter 1061, 1081

HCLASS parameter 1062, 1081

HEADALIGN attribute 744, 745, 749

headers and footers 1037, 1039
embedding field values 1039

heading code 377

HEADING command 1014

HEADING component 554, 556

headings 358, 359, 362, 372, 373, 374, 378,
548, 554, 556, 706, 714, 717, 762,
1037
aligning 714, 762
annotated text in 1037
creating 362, 1037
identifying in a style sheet 556
inserting data in 372, 373, 374, 378
limitations 359
positioning 706, 717

help reports 35

HewlettPackard plotters 1077

HGRID attribute 767, 772

hiding rows 1001, 1002

hiding sort field values 138

hiding sort values 138

hierarchies 975, 982, 989, 990

hierarchies in MASTER Files 989

high-order sort fields 884, 885, 887, 897, 899

high-resolution graphic devices 1029, 1030, 1076,
1077, 1078, 1081

Hewlett Packard plotters 1077, 1081

IBM devices and GDDM 1076, 1081

Tektronics terminals 1078, 1081

HISTOGRAM parameter 1047, 1081

histograms 1045, 1061, 1063, 1081

HAXIS 1061, 1081

VAXIS 1063, 1081

HMAX parameter 1061, 1081

HMIN parameter 1061, 1081

HOLD AT CLIENT command 423, 453

HOLD command 422, 423

HOLD file INTERNAL format 480

HOLD file keys and indexes 449

HOLD file structured 481

HOLD file suppressing field padding 478, 479

HOLD file text fields 472, 473

HOLD files 422, 423, 430, 443, 444, 481, 878,
1183
creating 1183

HOLD files and external sorting 155, 156

HOLD files and merge phrases 881, 882, 883

HOLD files and missing values 818

HOLD files for financial reports 1006

HOLD files text fields 455

HOLD format ALPHA 456

HOLD format DFIX 460

HOLD format DHTML 632

HOLD format INGRES 464

HOLD format INTERNAL 464, 478, 479

HOLD format PowerPoint 466

HOLD format Red Brick 466

- HOLD format SQL 467
- HOLD format SQLDBC 467
- HOLD format SQLINF 467
- HOLD format SQLMSS 467
- HOLD format SQLODBC 468
- HOLD format SQLORA 468
- HOLD format SQLSYB 468
- HOLD format SYLK 468
- HOLD format TAB 469
- HOLD format TABT 469
- HOLD format WP 470
- HOLD format XFOCUS 471
- HOLD formats 455
- HOLD formats EXL97 463
- HOLD formats FOCUS 463
- HOLD Master Files 423, 427, 434, 436, 437, 438, 439, 440, 441, 442
- HOLD option 685
- HOLDATTR command 434, 441
- HOLDATTR parameter 442
- HOLDLIST command 434, 439, 440
- horizontal axis features 1033, 1034, 1060, 1061, 1062, 1063, 1065
 - class and tick intervals 1062
 - grids 1065
 - scale 1063
 - sorting graph subjects 1033, 1034
 - width 1061
- horizontal bar charts 1048
- horizontal sort values 549, 551
 - identifying 551
- host fields 859
- host files 835, 843
- Hot Screen 81, 82, 85, 86, 87, 88, 89, 90, 92, 93, 95
 - activating 82
 - canceling commands 88
 - displaying BY fields with panels 90
 - function keys 92, 95
 - locating character strings 88
 - panel 93
 - previewing reports 90
 - printing 81
 - redisplaying reports 89
 - reissuing previous command 88
 - repeating commands 88
 - SAVE files 88
 - saving selected data 88
 - scrolling 85, 86, 87, 92
- Hot Screen commands 85, 86, 89, 94
 - BACK 86
 - BOTTOM 85
 - DOWN 85
 - FORW(ARD) 85
 - LEFT 86
 - NEXT 85
 - OFFLINE 94
 - OFFLINE CLOSE 94
 - RESET 86
 - RETYPE 89, 94
 - RIGHT 86
 - TOP 86
 - UP 86
- HSTACK parameter 1047, 1081
- HTICK parameter 1062, 1081
- HTML alignment 744
- HTML format 463
- HTML output 629
- HTML reports 523, 524, 796
 - fonts in 523, 524
 - hyperlinks in 796
- HTMLCSS SET parameter 612, 613

hyperlinks 621, 796, 797

I

ICU (Interactive Chart Utility) Interface 1033, 1074

identifying cells 965

identifying columns 959, 960, 961, 962

identifying columns by address 961

identifying columns by number 959

identifying columns by relative address 962

identifying columns by value 964

identifying columns in financial reports 958, 959

identifying contiguous columns 960

identifying groups of values 947

identifying ranges of multiple values 946

identifying report components 527, 529, 530, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 545, 549, 550, 551, 552, 554, 556, 558, 559, 560, 565, 566, 567

columns 529, 530, 532, 540

data 537, 538, 539

embedded fields 559, 560

free text 552

headings and footings 554, 556

page numbers 565, 566, 567

rows 533, 534, 541

skipped lines 565, 566, 567

sort values 551

text strings 558, 559

titles 549, 550, 551, 552

totals and subtotals 534, 535, 536, 542, 543, 545

underlines 565, 566

identifying rows 955, 956, 957

identifying rows in financial reports 956

IF command 1035

IF command with LIKE or UNLIKE 181

IF operator 171, 173

IF phrase 158, 181, 190, 191, 324

IF phrase expressions 324

IF/THEN/ELSE statements 354

IMAGE attribute 773, 774, 776, 780, 781

IMAGEALIGN attribute 774

IMAGEBREAK attribute 774

images 621, 773, 774, 776, 780, 781, 783, 784, 791, 792

adding to HTML reports 783

adding to PDF reports 784

adding to reports 773, 774, 776, 781

linking 791, 792

improving performance 155, 845, 903, 1110

IN phrase 398, 399, 400

IN-GROUPS-OF phrase 124, 125

IN-RANGES-OF phrase 124, 125

INCLUDES operator 187

indenting captions 981

independent paths 162, 164

index optimized retrieval 1109

INGRES formats 464

inheritance in style sheets 610, 627

inheriting attributes 581, 582

inline Style Sheets 607, 608

inner join 832, 848, 851

inner join structures 851

INSERT command 1111

INSERT INTO command 1099

inserting blank lines 565
inserting text in financial reports 968, 969
inserting underlines 565
inserting variables in free text 970
installing Pooled Tables 935
installing Pooled Tables on MVS 936
installing Pooled Tables on VM/CMS 936
inter-row calculations 953, 954
internal Cascading Style Sheets 607, 612, 620
INTERNAL format 464, 478, 479
internal matrixes 229, 911, 962
internal storage and field formats 333
INTERSECT operator 1103
irrelevant report data 807, 808
IS NOT operator 180, 181
IS operator 180, 181
ISO standard date-time formats 342
ITEM attribute 558
ITEMS data source 1149

J

JavaScript functions 788
 linking to 788
JOBFILE data source 1120, 1121
JOBHIST data source 1137
JOBLIST data source 1138
JOIN CLEAR command 874, 875
JOIN command 834, 839, 840, 843, 845, 856,
862, 1096, 1097, 1098

JOIN command and ALL parameter 835, 836
JOIN command and SQL Translator 1096, 1097,
1098
join structures 834, 843, 845, 853, 860, 1096,
1097, 1098
join structures and CHECK FILE command 871, 872
join structures and DBA security 835
join structures and DEFINE command 855, 856,
857, 858, 865, 866, 868
join structures and group fields 854
join structures and numeric data types 861
join structures and qualified field names 1098
join structures and virtual fields 855, 856, 857,
858, 866, 868
join structures and WHERE phrase 870
join types 832
joining data sources 36, 201, 834, 835, 843, 860
joining fields 860, 861
joins 832
justification regions 736
JUSTIFY attribute 732, 735, 737, 761
JUSTIFY syntax 755
justifying column titles 409, 410, 737, 738, 739,
740
justifying data 722, 733
justifying grand totals 740, 741
justifying headings and footings 734, 735, 736
justifying subtotals 740, 741

K

KEEPDEFINES parameter 865, 866
KEEPFILTERS SET parameter 201
key fields 429, 431
keyed retrieval 443, 444

L

LABEL attribute 533, 534, 552
LABELPROMPT attribute 801, 802
labels 801, 802, 803, 955
labels for rows 956, 957
lagging values 837
landscape orientation 513
last page number 554
last page number in a sort group 554
LE operator 176, 177, 352
LEDGER data source 1129
LEFT command in Hot Screen 86
left outer join 832, 851
left outer join structures 851
LEFTGAP attribute 707
LEFTMARGIN attribute 515, 516
LIKE operator 180, 181
limit FORECAST 244
limitations for headings and footings 359
limitations in display fields 59
limiting display fields 59
limits for display fields 39, 59

LINE attribute 556, 557
line termination characters 700
line-by-line formatting 755, 757
linear regression 260, 262
linear regression analysis 239, 241
linear regression analysis FORECAST 239, 241
linear regression equation 254, 256
linear regression equation FORECAST 254, 256
LINES SET parameter 730
linking report components 786, 787, 788
linking report pages 585, 797, 799
 heading text and 799
 images and 797
 in HTML reports 797
 page numbers and 799
 setting conditions 585
linking to external Cascading Style Sheets 617,
618, 621, 622, 623, 624
linking to JavaScript functions 788
linking with conditions 789, 790
LIST * command 49
LIST command 45, 47, 57, 98
list records 47
listing join structures 873
listing records 47, 48
literals 1104
LOAD CHART command 988, 989
load procedures 1113
loading a hierarchy into memory 988, 989
LOCATE command 88
locating character strings 88

LOCATOR data source 1139
LOCKED attribute 648
locking in excel spreadsheets 647
logical expression types 351
logical expressions 169, 179, 325, 351
logical operator 179
logical operators 169, 170, 171, 352
long field names 40, 41, 42
LOTUS format 464
low-resolution graphic devices 1029
LST prefix operator 69
LT operator 176, 177, 352

M

MACRO attribute 579, 580
macros 492, 578, 579, 580
 applying 579, 580
 defining 580
 overriding 580
mailing labels 801, 802, 803
maintaining across joins 201
maintaining filters 201
maintaining filters across joins 201
masked fields 180, 182
masking characters 180, 182, 947
masks 180, 181
Master Files 32, 178, 195, 977, 988, 1113
 hierarchies in 988
Master Files and MISSING attribute 810, 811, 812
Master Files for financial reports 977
Master Files for FML hierarchies 977, 988, 989

Master Files for hierarchies 977
Master Files HOLD files 423
MATCH command 878, 879, 880, 881, 1181
MATCH FILE command 879, 880, 881, 882, 883
MATCH FILE command and concatenated data sources 895
MATCH FILE command and display commands 888
MATCH FILE command and merge phrases 881, 882
MATCH FILE commands 878
matrix reports 115, 116, 270, 272
matrix type reports 115
matrixes 911, 912
MATRIXORDER attribute 801, 802
MAX prefix operator 64
MAX prefix operators 64
MAXEXTSRSTS 931
maximum prefix operators 64
measurement 613
measurement units 515
measuring for column width alignment 761
measuring for decimal alignment 761
medium-resolution graphic devices 1029, 1030, 1075, 1081
 Anderson Jacobson 1081
 Gencom 1081
merge phrases 881, 883
merge phrases and HOLD files 881, 882, 883
merge phrases and MATCH FILE command 881, 882

- merging data sources 878, 879, 880, 881, 882, 883, 884, 885, 887, 888, 895, 897, 899
 - merging data sources and display commands 888
 - merging data sources and PRINT command 888
 - merging data sources and SUM command 888
 - metrics files 687, 692, 693
 - METRICSFILE name 693
 - MIN prefix operator 64
 - MIN prefix operators 64
 - minimum prefix operators 64
 - MISSING attribute 178, 809, 815, 816, 817
 - MISSING attribute and extract files 818
 - MISSING attribute and Master Files 810, 811, 812
 - MISSING attribute and virtual fields 812
 - MISSING attribute limits 812
 - missing descendants 826, 827
 - missing instances 809
 - missing value data sources 807
 - missing values 178, 807, 808, 809, 810, 825, 826, 1069, 1081
 - GRAPH requests 1069, 1081
 - missing values and ALL parameter 826, 827
 - missing values and ALL prefix 825
 - missing values and DEFINE command 810, 811, 812, 813, 814
 - missing values and extract files 818
 - missing values and segment instances 809, 822, 823, 824
 - missing values and temporary fields 812
 - missing values for reformatted fields 78, 820
 - MODIFY procedure 753
 - MORE phrase 890, 891, 892, 895, 1036
 - MORE phrase and universal concatenation 891, 892
 - MOVIES data source 1148
 - multi-pane reports 801, 804
 - printing 804
 - multi-path data sources 51, 53, 218
 - multi-segment data sources 187
 - multi-segment files 187
 - multi-table HTML reports 730, 731
 - MULTILINES command 279, 280, 284, 367
 - multipath join structures 851
 - MULTIPATH parameter 162, 163, 164, 166
 - multiple display commands 139, 140
 - multiple display commands and ROW-TOTAL 274
 - multiple records 944
 - multiple sort fields 102, 103, 110, 139, 140
 - multiple values 944
 - multiple verbs 139
 - multiple virtual fields 215
 - multiple WHERE phrases 161
 - multiplication operator 327, 329
 - multivariate REGRESS 260, 261, 262
 - MVSMMSGDF utility 931
 - MVSMMSGSS utility 931
- N**
- naming CSS classes 610, 615
 - naming extract files 422

naming output files 422
National Language Support (NLS) 146
NE operator 178, 352
NEWPAGE command 365
NEXT command in Hot Screen 85
NLS (National Language Support) 146
NODATA character 808, 809, 828, 829
non-numeric fields 53, 54
non-recursive models 973
non-unique join structures 832, 835, 836
NOPAGE command 387
NOPRINT command 138, 391, 1001
NOSPLIT command 387, 388, 389
NOT FROM ... TO operator 175, 176
NOT LIKE operator 180
NOT operator 352
NOTOTAL command 319
numeric constants 1104
numeric data 124, 127, 640, 643
 unsupported formats 643
numeric data types 860
numeric data types and join structures 861
numeric expressions 325, 330
numeric fields 53
numeric functions 328
numeric operator expressions 327, 329

O

OBJECT attribute 558

OFFLINE CLOSE 94
OFFLINE command 81, 94
 TABLE 81
offline printing in Hot Screen 94
OMITS operator 179
ON GRAPH command 1072
ON phrase 411, 412, 413
ONLINE command 81
 TABLE 81
operators 327, 351
operators prefix 61
optimized join structures 1109
optimizing join structures 1109
optimizing sorting data 146
OR operator 169, 352, 945
order of evaluation 330
ORIENTATION attribute 510, 513
outer join 832
output file format 456
output file format DFIX 460
output file format HTML 463
output file format INGRES 464
output file format INTERNAL 464
output file format LOTUS 464
output file format PDF 465
output file format PDF OPEN/CLOSE 466
output file format PostScript (PS) 466
output file format PPT 466
output file format Red Brick 466
output file format SQL 467

- output file format SQLDBC 467
- output file format SQLINF 467
- output file format SQLMSS 467
- output file format SQLODBC 468
- output file format SQLORA 468
- output file format SQLSYB 468
- output file format SYLK 468
- output file format TAB 469
- output file format TABT 469
- output file format WP 470
- output file format XFOCUS 471
- output file formats 455, 462, 463
- output file formats EXL97 463
- output file formats FOCUS 463
- output file text fields 472, 473
- output files 422, 445
- output files and missing values 818
- output files text fields 455
- output format DHTML 632
- output formats 455
- OVER command 395, 396, 397, 718, 1045
 - GRAPH 1045

P

- parent and descendant components 493
- parent instances 826, 827
- parent segments in qualified field values 187
- partitioned FOCUS data sources 160
- PAUSE parameter 1075, 1081
 - GRAPH 1081
- PCHOLD command 422, 453
- PCHOLD files 453
- PCHOLD format 456
- PCHOLD format DFIX 460
- page count 554
- page fields 663
- page footings 363, 364
 - creating 363, 364
- page headings 359, 360, 361
 - creating 360, 361
- page margins 515, 516
 - attributes 515
- page numbers 380, 382, 383, 387, 554, 565, 566, 567
 - inserting 383
 - suppressing 387
- page orientation 510, 513
 - landscape 513
 - portrait 513
- PAGE parameter 387
- page size 510, 511
- PAGEBREAK command 381, 382
- PAGECOLOR attribute 510, 514
- PAGEFIELDS phrase 663, 664
- PAGEMATRIX attribute 801, 802
- PAGENUM component 565, 566, 567
- PAGESIZE attribute 510, 511
- panels 93

- padded fields 478, 479
- PAGE BREAK command 994
- page breaks 381, 382, 387, 388, 389
 - suppressing 388, 389
- page breaks in financial reports 994
- page colors 510, 514
 - setting 514

- PCHOLD format HTML 463
- PCHOLD format LOTUS 464
- PCHOLD format PDF 465
- PCHOLD format PDF OPEN/CLOSE 466
- PCHOLD format PS (PostScript) 466
- PCHOLD format TAB 469
- PCHOLD format TABT 469
- PCHOLD format WP 470
- PCHOLD formats 455, 462, 463
- PCHOLD formats EXL97 463
- PCT percent 64
- PCT prefix operators 64
- PCT.CNT prefix operator 66
- PDF (Portable Document Format) 465, 687, 688, 689, 693, 695
 - compound reports 689
 - font files 687
 - font map files 687
 - generating reports in 687
 - metrics files 687
 - PostScript Type 1 fonts 693
 - requirements 688
- PDF format on UNIX 699, 700
- PDF OPEN/CLOSE format 466
- PDF output 629
- PDFLINETERM parameter 699, 700
- percent (PCT) 64
- percentiles 127
- performance 146, 903, 1110
- performing calculations on dates 334
- PERSINFO data source 1140
- personal Cascading Style Sheets 627
- PICTURE RETRIEVE command 51
- pie charts 1053
- PIE parameter 1053, 1081
- PIVOT option 657
- PivotTables 657, 658, 660, 661, 663, 664
 - CACHEFIELDS phrase 661
 - generating 657
 - output 660
 - page fields 663
 - PAGEFIELDS phrase 663, 664
 - requests 657, 658
 - styling 658
 - TABLE syntax 660
- PLOT parameter 1081
- PLUS OTHERS 103
- PLUS OTHERS phrase 103
- POOL command 917
- POOLBATCH command 930
- Pooled Tables 916, 917, 922, 926, 932
 - memory management 926
 - subroutines for use with 922
 - trace facility 932
- Pooled Tables commands and sub pool boundaries 920
- Pooled Tables common selection criteria 928
- Pooled Tables configuration 936
- Pooled Tables example 918
- Pooled Tables FOCPARM parameters 936
- Pooled Tables FOCPOOLT 926
- Pooled Tables memory requirements 927
- Pooled Tables single TABLE clusters 921
- Pooled Tables sort selection 931
- Pooled Tables statistics 923

- Pooled Tables sub pool boundaries 919
- Pooled Tables temporary work file 926
- Pooled Tables use with batch requests 930
- Pooled Tables use with non-relational databases 928
- Pooled Tables use with relational databases 928
- POOLFEATURE 936
- pooling criteria 917
- POOLMEMORY 926
- POOLRESERVE 927
- Portable Document Format (PDF) 687, 688, 689, 693, 695
 - compound reports 689
 - font files 687
 - font map files 687
 - generating reports in 687
 - metrics files 687
 - PostScript Type 1 fonts 693
 - requirements 688
- portrait orientation 513
- POSITION attribute 706, 707, 708, 710, 711
- positional column referenced calculated values 227
- positional field references for COMPUTE command 227
- positional labels 955, 956
- positional referencing for columns 227
- positioning columns 707, 710, 711, 717
- positioning headings and footings 706, 708
- positioning report components 706, 707
- positioning text 374, 375
- POST command 1003
- posting data 1003, 1004
 - posting data in financial reports 1003
 - posting financial data 1002
- PostScript (PS) format 466, 508, 513, 687, 688, 689, 693
 - compound reports 689
 - font files 687
 - font map files 687
 - fonts 693
 - generating reports in 687
 - metrics files 687
 - PostScript Type 1 fonts 693
 - printing 513
 - requirements 688
- PostScript fonts 692
- PostScript Type1 fonts 692
- PowerPoint format 466
- PPT format 466
- precision 57
- predicting values with FORECAST 239
- prefix operators 60, 61, 62, 287, 288, 290, 302, 304, 305, 310, 313, 1033, 1034
 - GRAPH 1033, 1034
- prefix operators for display fields 60
- prefix operators in free-form reports 1015
- preserving field names 434
- preserving missing values 817, 818, 822
- preserving virtual fields 219
- PRINT * command 49
- PRINT command 45, 47, 49, 98, 888, 1029, 1081
 - GRAPH 1029, 1081
- PRINT command and merging data sources 888
- PRINT command unique segments 53
- printer/plotter selection for graphs 1029, 1081
- printing 801, 802, 803, 804
 - labels 801, 802, 803

printing (*continued*)
 multi-pane reports 804
PRINTONLY parameter 439, 440
PRINTPLUS parameter 83
procedures sorting data 146
PROD data source 1125
producing a direct percent of a count 66
PROTECTED attribute 648
protecting virtual fields 219, 220
PS (PostScript) format 508, 513, 687, 688, 689, 693
 compound reports 689
 font files 687
 font map files 687
 fonts 693
 generating reports in 687
 metrics files 687
 PostScript Type 1 fonts 693
 printing 513
 requirements 688
PS output 629

Q

qualified field names 40, 41, 42, 1098, 1103
qualified field names and SQL join structures 1098
qualified field names and SQL Translator 1103
qualified field values 113, 187
QUALTITLES parameter 408
query ? STAT command 147
query commands
 ? DEFINE 216
 ? STYLE 517
 ?F 43
 ?FF 44
querying HOLD files 423, 426

querying sort types 147
QUIT command 35, 1032, 1081
 in GRAPH request 1032, 1081
quotation marks 344
quote-delimited string 344, 345

R

range of values 946, 947
range tests 175, 176, 177
ranges 124, 125, 126, 127
RANKED BY phrase 122, 123
RANKED BY TOTAL phrase 122
ranking sort field values 73, 122, 123, 133, 134
reading selection values from a file 191, 192
reading values from a file 193
READLIMIT operator 189
READLIMIT relational operator 189
RECAP command 315, 316, 317, 318, 324, 938, 939, 954, 955
RECAP command and FML reports 953
RECAP command expressions 324
RECAP component 534, 542, 543, 547
RECAP expressions 955
RECAP rows 953
RECOMPUTE command 283, 284, 286, 287, 288, 302, 306, 307, 308, 310, 311, 313
RECOMPUTE command and propagation to grand total 296
RECOMPUTE prefix operators 313

- RECORDLIMIT operator 189, 190
- RECORDLIMIT relational operator 189, 190
- records 47, 159, 944
- records in multiple rows 948, 949
- recursive join structures 839, 840, 841, 1098
- recursive models 973, 974
- recursive structures 839, 840
- Red Brick format 466
- REDBRICK format 466
- ref_regress_usage 261
- reformatting fields 220, 222, 223
- refreshing external Cascading Style Sheets 626
- REGION data source 1131
- relational expressions 169, 351, 353
- relational operator 180
- relational operators 171, 173, 175, 176, 177, 178, 180, 182, 351
- relative column addresses 962
- relative starting positions 713
- renaming column totals 270, 273
- renaming HOLD files AS phrase 423
- renaming PCHOLD files 453
- renaming row totals 270, 273
- REPAGE command 381
- repeating fields 839
- repeating fields in join structures 839
- repeating rows 957
- repeating sort values 102
- REPLOTT command 1018
- report columns 76, 135
 - formatting 76
- REPORT component 529
- report components 493, 506, 518, 525, 527, 609, 614, 615, 617, 618, 706, 707, 785, 786, 787, 788
 - columns 527
 - linking 785, 786, 787, 788
 - positioning 706, 707
 - rows 527
- report footings 358, 363
 - creating 363
- report formatting 607, 610, 612, 614, 616, 617, 620, 625
 - external Cascading Style Sheets 614, 616, 617
 - inheritance in style sheets 610
 - internal Cascading Style Sheets 607, 612
 - tabular reports 617
- report headings 358, 359, 360, 361
 - creating 360, 361
- report output 36, 629
 - styled 629
- report panels 93
- report requests 36, 1089
 - creating 36
- report requests and SQL statements 1089
- report styling 605, 614, 616, 617, 620
 - external Cascading Style Sheets 614, 616, 617
- report SUM columns 135
- report syntax 492
- report titles 549, 553
- reporting against hierarchies 975, 978, 980, 981, 982, 990
- reporting commands 1177
- reporting options 1177
- reports 31, 32, 33, 34, 35, 36, 38, 40, 81, 82, 85, 89, 95, 357, 359, 363, 365, 374, 380, 381, 382, 383, 387, 391, 394, 395, 396, 398, 401, 402, 403, 404,

- reports (*continued*)
 - 405, 406, 408, 409, 410, 411, 419, 491, 492, 494, 495, 496, 499, 502, 518, 522, 523, 537, 538, 597, 635, 687, 689, 690, 723, 728, 730, 732, 796
 - applying sequential conditional formatting to 597
 - color values 522
 - comparing styled and non-styled 502
 - compound 689, 690
 - customizing 36, 38, 357, 499
 - displaying 36, 81, 82, 89, 95, 419, 496
 - displaying data 36
 - formatting 359, 363, 365, 374, 380, 381, 382, 383, 387, 391, 394, 395, 396, 398, 401, 402, 403, 404, 405, 406, 408, 409, 410, 411
 - formatting with Excel 2000 635
 - formatting with EXL97 635
 - formatting with style sheets 492, 495
 - formatting with StyleSheets 492, 494
 - identifying data 537, 538
 - in Portable Document Format (PDF) 687
 - in PostScript (PS) format 687
 - justifying columns 732
 - linking 796
 - positioning headers and footers 732
 - printing 36, 95
 - running 35
 - saving 36
 - scrolling in Hot Screen 85
 - selecting data 36
 - sorting data 36
 - specifying fields 36
 - specifying fonts for 523
 - styling 537, 538, 730
 - wrapping data 723
 - wrapping with alternative methods 728
- requirements for external sorting 146
- reserved words 1092
- RESET command in Hot Screen 86
- restricting sort field values 122, 123, 133, 134
- restrictions for distinct prefix operators 67, 68
- restrictions for DST prefix operators 68
- restructuring data 906
- retrieval data 162, 164
- retrieval limits 189, 190
- retrieval logic 904
- retrieval order 154, 155
- retrieving data 162, 164
- retrieving data values 943
- retrieving financial data 1002
- retrieving financial data values for rows 942, 943
- retrieving multiple values with masking characters 947
- retrieving posted data 1004, 1005
- retrieving records 69, 189, 190, 845
- retrieving values for rows 942
- retrieving values with masking characters 947
- returned fields 335
- RETYPE command 89, 94
- reusing FOR field values 948
- reusing output reports 421
- reusing records 948, 949
- reusing report output 421
- reusing tag values 942, 948, 949, 982
- reusing values 949
- RIGHT command in Hot Screen 86
- RIGHTGAP attribute 707
- RIGHTMARGIN attribute 515, 516
- RNK. prefix operator 73

- rotating data sources 904
 - rounding numeric values 327
 - row formatting 992
 - row labels 957
 - row percent (RPCT) 64
 - row titles 552, 990, 991, 992
 - row totals 270, 272, 273, 276, 277
 - ROW-TOTAL phrase 270, 271
 - ROW-TOTAL with ACROSS and multiple display commands 274
 - rows 533, 534, 541
 - identifying in a style sheet 541
 - rows and calculations 953, 954, 955
 - rows in financial reports 941
 - ROWTOTAL attribute 536
 - RPCT prefix operator 64
 - RPCT row percent 64
 - rules in Cascading Style Sheets 609, 618, 625
 - RUN command 35
- S**
- SALES data source 1123, 1124
 - SALHIST data source 1141
 - SAME DB 445
 - SAME_DB extract files 445, 446, 448, 449
 - SAME_DB HOLD files 445, 446, 448, 449
 - SAME_DB HOLD format 445, 446, 448
 - SAME_DB HOLD format columns 448
 - SAME_DB output files 445, 446, 448, 449
 - sample data sources 1113, 1116, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1129, 1130, 1131, 1132, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1144, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1160, 1161, 1162, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172
 - CAR 1126, 1127
 - CENTGL 1169, 1170
 - CENTSTMT 1171, 1172
 - CENTSYSF 1170
 - Century Corp 1158, 1160, 1161, 1162, 1164, 1165, 1166, 1167, 1168, 1169
 - COMASTER Master File 1144
 - COURSE 1136, 1137
 - COURSES 1132
 - EDUCFILE 1121, 1122
 - EMPLOYEE 1116, 1118, 1119
 - FINANCE 1130
 - Gotham Grinds 1152, 1153, 1154, 1155, 1156, 1157
 - ITEMS 1149
 - JOBFILE 1120, 1121
 - JOBHIST 1137
 - JOBLIST 1138
 - LEDGER 1129
 - LOCATOR 1139
 - MOVIES 1148
 - PERSINFO 1140
 - PROD 1125
 - REGION 1131
 - SALES 1123, 1124
 - SALHIST 1141
 - TRAINING 1135, 1136, 1142
 - VIDEOTR2 1150, 1151
 - VideoTrk 1146, 1147
 - SAVB command 449
 - SAVB files 449
 - SAVE command 422, 449

- SAVE files 88, 449, 1072
 - GRAPH 1072
 - in Hot Screen 88
- SAVE format 456
- SAVE format EXL2K 462
- SAVE format HTML 463
- SAVE format LOTUS 464
- SAVE format PDF 465
- SAVE format SYLK 468
- SAVE format TAB 469
- SAVE format TABT 469
- SAVE format WP 470
- SAVE formats 455, 463
- SAVEMATRIX parameter 911, 912
- saving intermediate report results 1002
- saving output files 422
- saving report output 421, 422
- saving reports 421, 422
- saving rows 1003, 1004
- saving selected data 88
- saving virtual fields 865, 866, 868
- scalar functions 1104
- scatter diagrams 1056
- SCREEN parameter 82
- screening conditions 195
- screening segments 870
- screening values 229
- scrolling in Hot Screen 85
- SEG. operator 43
- segment instances 807, 826, 827
 - segment instances and missing values 822, 823, 824
 - segment locations 217
 - segment types 69, 70
 - segments 43, 162, 166, 809
 - SEGTYPE parameter 443
 - selecting records 157, 158, 162, 164, 167, 168, 169, 170, 171, 173, 174, 179, 180, 187, 188, 190, 191, 192, 193, 194, 204, 1016
 - selecting records with IF phrase 192, 194
 - selecting records with VSAM 204
 - selecting sort procedures 147
 - selecting sort types 147
 - selecting values using WHERE phrase 192
 - selecting values with IF phrase 194
 - selection criteria 157, 159, 160, 161, 162, 164, 171, 173, 174, 191, 192, 193, 194, 320
 - selection values 193
 - selection values with IF phrase 194
- SEQUENCE attribute 718
- sequential conditional formatting 586, 597
- SET ACROSSPRT 111
- SET ALL parameter 826, 827
- SET ASNAMES parameter 434
- SET AUTOINDEX parameter 907
- SET AUTOPATH parameter 907
- SET BLANKINDENT parameter 997, 998
- SET CNOTATION parameter 229, 230, 962, 963
- SET COMPMISS parameter 820

- SET COMPUTE = NEW command 912
- SET COMPUTE command 915
- SET COUNTWIDTH parameter 57
- SET DATEFORMAT parameter 337
- SET DEFINES command 915
- SET DUPLICATECOL command 141
- SET END command in GRAPH 1081
- SET ESTLINES parameter 925
- SET ESTRECORDS parameter 925
- SET EXTSORT parameter 146, 147
- SET FILTER parameter 195, 198
- SET FORMULTIPLE parameter 948, 949
- SET HOLDATTR parameter 434, 441, 442
- SET HOLDLIST parameter 434, 439, 440
- SET HOLDMISS parameter 818
- SET KEEPFILTER parameter 201
- SET MAXEXTSRTS parameter 931
- SET parameter ACROSSLINE 108
- SET parameter ESTLINES 148
- SET parameter ESTRECORDS 148
- SET parameter EXTAGGR 153
- SET parameter EXTHOLD 156
- SET parameter for BYDISPLAY 102
- SET parameter NULL=ON 470
- SET parameters 34, 41, 82, 83, 87, 90, 93, 108, 387, 405, 408, 410, 411, 419, 505, 612, 613, 630, 699, 700, 718, 728, 729, 730, 907, 999, 1078, 1081
 - BYPANEL 90
 - BYSCROLL 87
 - COLUMNS 93
 - EMPTYREPORT 419
- SET parameters (*continued*)
 - FIELDNAME 41
 - FILE 34
 - GRAPH 1081
 - GRID 728
 - HTMLCSS 612, 613
 - LINES 730
 - PAGE 387
 - PANEL 93
 - PDFLINETERM 699, 700
 - PRINTPLUS 83
 - QUALTITLES 408
 - SCREEN 82
 - SHOWBLANKS 630
 - SPACES 405, 718
 - SQUEEZE 728
 - STYLEMODE 730
 - STYLESHEET (STYLE) 505
 - UNITS 612, 613
 - XRETRIEVAL 90
- SET parameters and EXTHOLD 156
- SET POOL parameter 917
- SET POOL=OFF command 917
- SET POOL=ON command 917
- SET POOLBATCH parameter 930
- SET POOLFEATURE parameter 936
- SET POOLMEMORY parameter 926
- SET POOLRESERVE parameter 927
- SET PSPAGESETUP command 510, 513
- SET SAVEMATRIX parameter 911, 912
- SET SORTLIB parameter 931
- SET SQLTOPTTF parameter 1110
- SET SQUEEZE parameter 728
- SET SUMMARYLINES parameter 296
- SET SUMMARYLINES parameter 295, 297, 307
- SET TARGETFRAME command 794

- SET TRACEOFF parameter 932
- SET TRACEON parameter 932
- SET WPMINWIDTH 471
- setting retrieval order 155
- sheet names 549
- short path definitions 824
- SHOWBLANKS SET parameter 630
- simple moving average 239, 241, 245, 247
- simple moving average FORECAST 241, 245, 247
- SIZE attribute 781
- SKIP-LINE command 401, 402, 403
- SKIPLINE component 565, 566, 567, 574
- skipped lines 565, 566, 567, 574
 - formatting 574
- sort field values 133
- sort fields 97, 950
- sort fields for multi-path data sources 99
- sort fields using multi-path data sources 107
- sort footings
 - limitations 359
- sort headings 359, 758, 762
 - aligning 762
 - limitations 359
- sort multiple fields 102, 110
- sort order 102, 110, 124, 125, 126, 127
- sort phrases 47
- sort selection with Pooled Tables 931
- sort sequence 99, 107, 950
- sort temporary fields 99, 107
- sort values 98, 99, 124
- sorting by calculated values 135, 136, 137
- sorting by columns 107, 108, 110, 137
- sorting by rows 99
- sorting columns 135, 136
- sorting columns by 107
- sorting data 103, 146, 147
- sorting data by columns 107, 108
- sorting data by multiple fields 102, 110
- sorting data by rows 99, 100, 103
- sorting report columns 135, 136
- sorting reports 97, 98, 1016
- sorting reports by calculated values 229
- sorting rows by 99, 100, 102, 103
- sorting with COMPUTE command 137
- SORTLIB 931
- SORTWORK files 148
- SPACES parameter 405
- SPACES SET parameter 718
- spacing between columns 717
- specifying date-time values 337
- specifying fields in reports 36
- specifying indentation between levels 997
- specifying ranges in financial reports 947
- specifying sort order 116, 117, 118, 119, 120, 121
- specifying URLs 776
- splits 387, 388, 389
 - preventing 388, 389
- spot markers 374, 376, 559
 - OX 376
- SQL format 467
- SQL join structures 1096, 1097, 1098

- SQL join structures and qualified field names 1098
- SQL SELECT statement 1095
- SQL statements 1089, 1090, 1092
- SQL statements and FOCUS TABLE requests 1089
- SQL Translation Services 1090, 1092
- SQL Translator 1089
- SQL Translator and aliases 1098
- SQL Translator and Cartesian product answer sets 1102
- SQL Translator and Continental Decimal Notation (CDN) 1102, 1103
- SQL Translator and CREATE TABLE command 1099, 1100
- SQL Translator and CREATE VIEW command 1100, 1101
- SQL Translator and date formats 1104, 1105
- SQL Translator and date-time values 1106, 1107, 1108
- SQL Translator and DELETE command 1111
- SQL Translator and DROP VIEW command 1100, 1101
- SQL Translator and expressions 1104
- SQL Translator and field names 1103
- SQL Translator and index optimized retrieval 1109
- SQL Translator and INSERT command 1111
- SQL Translator and INSERT INTO command 1099
- SQL Translator and JOIN command 1096, 1097, 1098
- SQL Translator and join structures 1109
- SQL Translator and reserved words 1092
- SQL Translator and SQLTOPTTF parameter 1110
- SQL Translator and time and timestamp fields 1104, 1105
- SQL Translator and UPDATE command 1111
- SQL Translator commands 1093, 1095
- SQL Translator commands and formatting commands 1094
- SQL Translator and INSERT INTO command 1099
- SQLDBC format 467
- SQLINF format 467
- SQLMSS formats 467
- SQLODBC formats 468
- SQLORA formats 468
- SQLSYB formats 468
- SQLTOPTTF parameter 1110
- SQUEEZE attribute 644, 718, 719, 720, 729, 730
- stacking columns 717
- STAT query 147
- structure diagrams 1113
- structured HOLD files 481
- STYLE attribute 519
- STYLE parameter 505
- style sheet attributes 617
- Style Sheet files 494
- style sheets 495, 527, 537, 548, 605, 606, 610, 612, 613, 616, 620, 625, 626, 627, 695, 992
 - Cascading Style Sheets (CSS) 606, 612, 613
 - declarations 695
 - FOCUS StyleSheets 620

- style sheets (*continued*)
 - inheritance 610, 627
 - refreshing external Cascading Style Sheets 626
 - selecting 495
 - troubleshooting 626
- styled output formats 629
- styled reports 502
- STYLEMODE SET parameter 730
- StyleSheet attributes 614, 615, 617, 618, 621, 623, 624, 992
 - CLASS 614, 615, 617, 618
 - CSSURL 623, 624
- StyleSheet declarations 500, 501, 506, 507, 785
 - attribute=value pairs 506
 - defining links with 785
 - improving readability 507
- StyleSheet syntax 506, 508
- StyleSheets 491, 492, 495, 498, 499, 505, 506, 507, 508, 518, 525, 526, 529, 578, 579, 580, 581, 582, 585, 587, 588, 589, 590, 599, 620, 688, 705, 732, 796, 801, 804
 - ACROSSCOLUMN attribute 599
 - activating 505
 - attribute inheritance 581, 582
 - attributes 518
 - CHECK STYLE command 508
 - comments 507
 - conditional styling and 585, 587, 588, 589, 590
 - customizing 498, 499
 - declarations 578
 - FOCSTYLE file 506
 - identifying report components 526
 - linking reports with 796
 - macros 578, 579, 580
 - multi-pane reports and 801, 804
 - positioning components 732
 - printing 508
 - REPORT component 529
 - report components 525
 - requirements 688
- StyleSheets (*continued*)
 - sample files 498
 - syntax 506
 - WHEN command 599
- styling reports 605, 614, 615, 616, 617, 620
 - external Cascading Style Sheets 614, 616, 617
 - free-form reports 614, 615
- sub pools 919
- sub pools boundaries 919
- sub pools pooling restrictions 919
- sub-total calculated values 284
- SUB-TOTAL command 279, 280, 281, 282, 287, 288, 290, 304, 305, 306, 307, 309, 310, 311, 313
- SUB-TOTAL command and propagation to grand total 296
- SUB-TOTAL prefix operators 313
- subcomponents 527
- SUBFOOT command 367, 368
- SUBFOOT component 554, 556
- subfootings 367, 368, 372, 378, 379, 414, 416
 - creating 367
 - displaying 414, 416
 - inserting data in 372, 378, 379
- SUBHEAD component 554, 556, 777
 - embedding images in 777
- subheadings 365, 366, 372, 378, 415
 - creating 365
 - displaying 415
 - inserting data in 372, 378
- subroutines 966
- subroutines and Pooled Tables 922
- subtotal calculated values 284

- SUBTOTAL command 279, 280, 281, 282, 287, 288, 290, 304, 305, 306, 308, 309, 310, 311, 313
 - SUBTOTAL command and propagation to grand total 296
 - SUBTOTAL component 534, 542, 543, 545
 - SUBTOTAL prefix operators 313
 - SUBTOTAL SUMMARYLINES command 307
 - subtotals 283, 284, 285, 286, 306, 307, 315, 316, 317, 318, 414, 534, 535, 536, 542, 543, 544, 545, 547
 - displaying 414
 - identifying in a style sheet 544, 545, 547
 - subtraction operator 329
 - SUM command 45, 53, 54, 98, 888
 - SUM command and merging data sources 888
 - SUM prefix operator 71
 - SUMMARIZE command 283, 284, 285, 287, 288, 302, 304, 305, 306, 307, 313
 - SUMMARIZE command and propagation to grand total 296
 - SUMMARIZE prefix operators 313
 - summary commands 306, 310, 312
 - summary line processing 295
 - summary lines 320, 321
 - summary values 287, 288, 313
 - SUMMARYLINES SET parameter 295, 296, 297, 307
 - summing columns 135, 136
 - summing field values 98
 - summing report columns 135, 136
 - summing values 98
 - SUMPREFIX parameter 155
 - SUP-PRINT command 138
 - supplying data directly in FML 952, 953
 - supported data sources 834
 - suppressing display in financial reports 1001, 1002
 - suppressing field display 391, 392
 - suppressing field padding 478, 479
 - suppressing grand totals 319
 - suppressing rows 1001, 1002
 - suppressing rows in financial reports 1001, 1002
 - suppressing sort field values 138
 - SUPPRINT command 391, 392
 - SYLK format 468
 - syn_regress_mult 260
 - SYNCSORT 931
 - SyncSort utility 146, 147
 - system variables
 - TABPAGENO 382
- ## T
- TAB format 469
 - tab-delimited output files 469
 - TABFOOTING component 554, 556
 - TABHEADING component 554, 556
 - TABLASTPAGE system variable 554, 561
 - TABLE command 81, 88, 89, 90, 94, 95, 1178
 - displaying reports in Hot Screen 81
 - displaying reports in TOE 95
 - displaying reports with parameter set to ONLINE 81

- TABLE command (*continued*)
 - extracting data 95
 - extracting data from Hot Screen 88
 - OFFLINE 81, 94
 - ONLINE 81
 - previewing reports 89, 90
 - printing reports 81
 - redefining field formats 89
 - redisplaying reports 89, 94
- TABLE FILE command 33, 34, 36
- TABLE language formatting 622
- TABLE requests 1089
- TABLE syntax 660
- TABLEF command 910, 1110, 1180
- TABLEF command and data retrieval 910, 911
- TABLEF command and SQL Translator 1110
- TABPAGENO variable 382, 383
- TABT format 469
- tabular reports 614, 615, 617, 626
- tag names 840
- TAG rows 1001, 1002
- TAG rows suppressing display in financial reports 1001, 1002
- tag values 942
- target frames 794, 795, 796
 - specifying 794, 795
- TARGETFRAME SET parameter 796
- TD element 609
- Tektronics terminals 1078
- temporary field types 206
- temporary fields 36, 206, 208, 209, 224
 - calculated values 224
 - creating 36
- temporary fields and missing values 812
- temporary sort fields 107
- temporary tables 445, 448, 449
- temporary tables extract files 446
- temporary tables HOLD files 448, 449
- temporary tables output files 446, 448, 449
- Terminal Operator Environment (TOE) 95
- TERMINAL parameter 1081
- testing character strings 179, 180, 181, 182, 183, 184, 185, 186
- testing data fields 180, 181
- testing for blanks or zeros 817
- testing for existing data 178, 816
- testing for missing segment instances 827, 828
- testing for missing values 815, 816, 827, 828
- testing multi-segment files 187
- text 374, 375
 - positioning 375
- TEXT component 558
- text field output files 472, 473
- text fields 99, 753, 754, 755
 - aligning 753, 755
 - formatting 753
 - styling 754
- text fields and alphanumeric fields 263
- text fields in DEFINE and COMPUTE 263
- text fields output files 455
- text rows 968, 970
- text strings 559
- tick intervals in GRAPH 1062
- TILE column 127, 129, 132
- tile fields 127, 129, 132, 133

TILES phrase 127, 129, 132, 133

time fields 1104

time fields and SQL Translator 1105

timestamp data type 337

timestamp fields 1104

timestamp fields and SQL Translator 1105

TITLE attribute 441, 442

TITLE component 550, 551

titles 548, 549

TITLETEXT attribute 553

TO phrase 946, 947

TOP command in Hot Screen 86

TOPGAP attribute 707, 715, 716

TOPMARGIN attribute 515, 516

TOT prefix operator 71

total page count 554

totals 270, 276, 277, 278, 534, 535, 536, 544, 545, 547
 identifying in a style sheet 544, 545, 547

trace facility for Pooled Tables 932

TRACEOFF command 932

TRACEON command 932

TRAINING data source 1135, 1136, 1142

transferring files using FTP 684

treating as literal masking characters 184, 185

treating as literal wildcard characters 184, 185

treating literal masking characters 186

treating literal wildcard characters 186

triple exponential smoothing 252, 254

triple exponential smoothing FORECAST 252, 254

troubleshooting Cascading Style Sheets 626

TYPE attribute 526, 529, 783
 in StyleSheets 526
 REPORT component 529

U

UNDERLINE command 403, 404

UNDERLINE component 565, 566, 574, 575

underlines 565, 566, 576, 577
 adding 576, 577
 deleting 576, 577

underlines in financial reports 993

underlining values 574, 575

Uniform Resource Locators (URLs) 776, 786, 787, 793, 794
 linking to 786, 787
 specifying 793, 794

UNION operator 1103

unique join structures 832, 835, 836, 848

unique segments 49, 56

unique segments for PRINT command 51

UNITS attribute 515, 612, 613

units of measurement 515, 613

UNITS parameter 612, 613

universal concatenation 890, 1036
 MORE phrase 1036

universal concatenation and field names 893, 894

universal concatenation and MORE phrase 890, 891, 892

UNIX, PDF files for 699

UNLIKE 181

unsupported date and numeric formats 643

- UP
 - command in Hot Screen 86
 - UPDATE command 1111
 - URLs (Uniform Resource Locators) 776, 786, 787, 793, 794
 - linking to 786, 787
 - specifying 793, 794
 - user-coded programs 1183
 - using concatenation with AnV fields 347
 - using CONTAINS and OMITS with AnV fields 348
 - using EDIT function with AnV fields 348
 - using LIKE fields with AnV fields 348
 - using operators with AnV fields 349
- V**
- value dates 332
 - value format for dates 333
 - values 372, 945
 - embedding 372
 - values for columns 964
 - values in multiple rows 948, 949
 - varchar fields 461
 - variable length character expressions 347
 - VAUTO parameter 1063, 1081
 - VAXIS parameter 1063, 1081
 - VCLASS parameter 1064, 1081
 - verbs 45, 98
 - VERBSET attribute 143
 - verifying external sorting 147
 - vertical axis features 1026, 1057, 1062, 1063, 1064, 1065, 1081
 - class and tick intervals 1062, 1064, 1081
 - graph element 1026
 - vertical axis features (*continued*)
 - grids 1026, 1057, 1065, 1081
 - height 1063
 - scale 1063
 - vertical spacing 715
 - VGRID attribute 767, 772
 - VGRID parameter 1057, 1065, 1081
 - VIDEOTR2 data source 1150, 1151
 - VideoTrk data source 1146, 1147
 - virtual fields 195, 196, 206, 208, 209, 217, 218
 - virtual fields and join structures 855, 856, 857, 858, 865, 866, 868
 - virtual fields and MISSING attribute 812
 - virtual fields and missing values 812
 - VMAX parameter 1063, 1081
 - VMIN parameter 1063, 1081
 - VMSORT utility 147, 931
 - VSAM data sources 204
 - VSAM record selection efficiencies 204
 - VTICK parameter 1064, 1081
 - VZERO parameter 1069, 1081
- W**
- Web browser support for Cascading Style Sheets 607, 625, 626
 - WHEN attribute 585, 587
 - WHEN clause 411, 412, 413, 418
 - WHEN command 599
 - WHEN EXISTS phrase 1002
 - WHEN phrase 280, 320, 324

- WHEN phrase expressions 324
 - WHEN=FORECAST attribute 244
 - WHERE operator 171, 173, 176, 177
 - WHERE phrase 158, 159, 160, 161, 167, 169, 180, 182, 191, 192, 193, 324
 - WHERE phrase and existing data 816
 - WHERE phrase and join structures 870
 - WHERE phrase and missing values 815, 816, 817
 - WHERE phrase expressions 324
 - WHERE TOTAL phrase 167, 168, 169
 - WHERE-based join structures 832
 - widow lines 387, 388, 389
 - preventing 388
 - WIDTH attribute 755, 761
 - measuring 761
 - width of columns 719, 720, 721
 - WIDTH syntax 755
 - wildcard characters 180, 182
 - window titles 549
 - WITH CHILDREN parameter 978
 - WITHIN phrase 113
 - worksheet titles 549
 - WP format 470
 - WPMINWIDTH parameter 471
 - WRAP attribute 723, 730
 - WRAP parameter 723
 - wrapgap StyleSheet attribute 725
 - wrapping data 722, 723, 724, 728
 - alternative methods 728
 - by Web browser functionality 724
 - using a SET command 724
 - WRITE command 53
- ## X
- XFOCUS format 471
- ## Y
- Y2K attributes in Master Files 337
 - Year 2000 attributes in Master Files 337
 - YRTHRESH attribute 337
- ## Z
- z/OS requirements 147
 - zero records in a report 419
 - zeros 817

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

Web form: <http://www.informationbuilders.com/bookstore/derf.html>

Name: _____

Company: _____

Address: _____

Telephone: _____ Date: _____

Email: _____

Comments:

FOCUS for Mainframe

Creating Reports

Version 7.6

