FOCUS for Mainframe
Version 7.3

New Features

Introducing ...

XFOCUS Database, MDI, FML Hierarchies, Excel 2000, Summary Prefix Operators, Wide Files, and IEDIT

Information Builders

Contents

7.3 New Features

NF771: Long Dialogue Manager Variables
- Using Long Strings With -SET

NF830: Minus Edit Format Option

NF861: Increased Row Size Limit for Relational Data Adapters

NF866: DB2 V7.1 FETCH FIRST n ROWS Support

NF888: Excel 2000
- Creating Styled Excel 2000 Reports
- National Language Support with EXL2K
- Displaying Formatted Dates and Numeric Values
- Controlling Column Width and Wrapping
- Using the Excel 2000 Formula Option
- Using the Excel 2000 PIVOT Option
- Designating CACHEFIELDS in PivotTables
- Designating PAGEFIELDS in PivotTables
- Transferring Excel 2000 Formatted Files Using FTP

NF916: XFOCUS Data Source
- Accessing an XFOCUS Data Source

NF917: Long Segment Names

NF920: NORMSDST and NORMSINV Subroutines
- Characteristics of the Normal Distribution

NF922: AnV Format
Propagating an AnV Field to a HOLD File
Using AnV Fields in Expressions
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
Using AnV Fields With Functions
TRIMV: Removing Characters From a String
LENV: Returning the Length of an Alphanumeric Field
LOCASV: Creating a Variable Length Lower Case String
POSITV: Finding the Beginning of a Variable Length Substring
SUBSTV: Extracting a Variable Length Substring
UPCASV: Creating a Variable Length Upper Case String
Inputting AnV Fields With the MODIFY FIXFORM Command

NF923: COMBINE Command Enhancement
COMBINE Command Support

NF926: Controlling Report Page Numbering

NF928: Compiled DEFINEs

NF929: Multivariate Linear Regression

NF930: Holding Missing Values

NF931: Multiple FOLD-LINE

NF932: Including Data in Multiple FOR Rows in FML

NF935: Long Index Names

NF936: Summary Prefix Operators
- Controlling Summary Line Processing
- Using Prefix Operators With Calculated Values
- Using Multiple SUB-TOTAL or SUMMARIZE Commands

NF937: Optimizing Selection of Relational Variable Length Character Data Types

NF938: Decimal Alignment of Headings
- Aligning Decimals in a Multi-Line Heading or Footing

NF939: SET SAVEDMASTERS

NF940: SET HNODATA

NF943: Increased Number of ACROSS Columns

NF944: Increased Space for Column Titles

NF945: Creating Compound PDF or PostScript Reports

NF946: &FOCUSER

NF951: Defining and Displaying FML Hierarchies
- Requirements for FML Hierarchies
- Displaying an FML Hierarchy
- Consolidating an FML Hierarchy
- Loading a Hierarchy Manually

NF952: TABLASTPAGE

NF953: File Extensions For Maintain Procedures

NF954: Enhanced Screening Conditions for Maintain

NF955: Long Qualified Field Names

NF956: Wide Lines

NF957: FIELDTYPE=R

NF958: StyleSheet Enhancements
- Introduction to Styled Reports
  - How to Choose a Type of Style Sheet
- Choosing an Output Format
- Styling Reports With StyleSheets
  - 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
- Creating a Styled Report
- Styling the Page Layout
  - Selecting Page Size, Orientation, and Color
  - Setting Page Margins
  - Displaying Current Settings: The ? STYLE Query
- Specifying Font Format in a Report
  - Specifying Fonts for Reports

Reusing FOCUS StyleSheet Declarations With Macros
Defining a FOCUS StyleSheet Macro
Applying a FOCUS StyleSheet Macro
Identifying Report Components
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
Positioning a Report Component
Arranging Pages and Columns on a Page
Determining Column Width
Wrapping and Justifying Report Components
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
Aligning Heading and Footing Elements
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

FOCUS StyleSheet Attribute Inheritance
Adding Grids and Borders
Conditionally Formatting in a StyleSheet
Applying Sequential Conditional Formatting
Using Conditional Grid Formatting in a Field
Adding an Image to a Report
Linking in a Report
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
Working With Mailing Labels and Multi-Pane Pages
Working With PostScript and PDF Reports
Adding PostScript Type 1 Fonts for PS and PDF Formats

NF959: DATASET for a Segment in a Master File
DATASET Behavior in FOCUS Data Sources

NF961: USERFCHK Setting
Prerequisites for Function Parameter Verification

NF964: Extensions to FORECAST
Using Exponential Smoothing
Using Single Exponential Smoothing
Using Double Exponential Smoothing
Using Triple Exponential Smoothing
Using a Linear Regression Equation
Graphing a FORECAST Request

NF965: Raised Limit for EXTERNAL INDEX

NF968: FMLINFO Function

NF970: SUFFIX = TAB
Accessing a SUFFIX=TAB Data Source
Creating a Tab-Delimited Data Source

NF971: SET NULL=ON

NF972: NEWPAGE

NF973: Cascading Style Sheets
- What Are Cascading Style Sheets?
- The Notion of Browser Dependence
- Types of Cascading Style Sheets
- Cascading Style Sheets and Precedence Rules
- Cascading Style Sheet Formatting Statements: Rules and Classes
- Selecting a CSS Rule
- Naming CSS Classes
- Inheritance and CSS
- Generating an Internal Cascading Style Sheet
- Selecting a Unit of Measurement
- Working With External Cascading Style Sheets
- Applying CSS Styles
- Using an External CSS - A Graphical Overview
- Combining CSS Styling With Other Formatting Methods
- Combining an External CSS With a FOCUS StyleSheet
- Combining an External CSS With a TABLE Language Formatting
- Linking to an External Cascading Style Sheet
- FAQ About Using External Cascading Style Sheets
- Troubleshooting Cascading Style Sheets

NF977: IN-RANGES-OF

NF979: SET HOLDFORMAT

NF980: SET BLANKINDENT

NF981: Increased Number of Sort Headings and Footings

NF982: SET BYDISPLAY

NF983: Indenting FML Reports

NF984: IEDIT Command
- IEDIT Facilities on CMS
- Installing IEDIT on MVS
- Installing IEDIT on CMS
- Using IEDIT on CMS
- Using IEDIT on MVS

NF985: DYNAM Support for New GDG Iterations
- Using DYNAM With Relative GDG Numbers

NF986: GROUP Fields in FOCUS Data Sources

NF987: MDI for FOCUS and XFOCUS Data Sources
- Specifying an MDI in the Access File
- Creating a Multi-Dimensional Index
- Choosing Dimensions for Your Index
- Building and Maintaining a Multi-Dimensional Index
- Using a Multi-Dimensional Index in a Query
- Querying a Multi-Dimensional Index
- Using AUTOINDEX to Choose an MDI
- Analyzing a Retrieval Request With AUTOINDEX
- Joining to a Multi-Dimensional Index
- Choosing Source Fields
- Encoding Values in a Multi-Dimensional Index
- Partitioning a Multi-Dimensional Index
- Querying the Progress of a Multi-Dimensional Index
- Displaying a Warning Message

NF989: Excel 97
Creating Styled Excel 97 Reports

NF994: FOCUS SETS From Maintain

NF995: Raised Limit for IF-THEN-ELSE Conditions

7.2 New Features

NF551: AUTOIMS
AUTOIMS Prerequisites
Installing AUTOIMS
Invoking AUTOIMS
AUTOIMS Master File Generation Window
COBOL FD Translator Options Window
Completing AUTOIMS Processing
Accessing the Generated Master File and FOCPSB
Generated Master File

NF646: WHERE-Based JOIN
Conditional JOIN Processing
Preserving Virtual Fields During Join Parsing
Preserving Virtual Fields Using the KEEPDEFINES Parameter
Preserving Virtual Fields Using DEFINE FILE SAVE and RETURN
Clearing Joins
Listing JOIN Structures: The ? JOIN Query

NF767: Long Alphanumeric Fields

NF787: Increased Report Width

NF788: Tiles
Grouping Data in Tiles

NF789: FORECAST
FORECAST Processing
Forecasting Methods
Using a Simple Moving Average
Using an Exponential Moving Average
Using a Linear Regression Equation
FORECAST Reporting Techniques

NF790: ACROSS-TOTAL

NF792: SET CDN Command Enhancement

NF793: Displaying Fractions

NF796: Unlimited Nested -INCLUDE Commands

NF797: Field-Based Reformatting
Using Field-Based Formatting

NF801: DB2 Data Adapter SET SSID Command for MSO

NF802: Long Master File Names
Using Long Names on OS/390
Member Names for Long Master File Names in OS/390
Creating a HOLD Master File With a Long Name on OS/390
Creating RDBMS Tables With Long Master File Names

NF817: Creating Comma-Delimited Files
Types of Comma-Delimited Output Files

NF824: STRIP Subroutine

NF825: SET PCOMMA

NF826: SQUEEZ Subroutine

NF827: TRIM Subroutine

NF828: SUFFIX = COM/COMT/TABT
- Accessing SUFFIX=COM Data Sources
- Accessing SUFFIX=COMT Data Sources
- Accessing SUFFIX=TABT Data Sources

NF829: AUTODATE for FOCUS Data Sources
- Creating and Using the AUTODATE Field

NF833: Creating Tab-Delimited Files

NF880: Exit on Error
- Controlling Batch Error Processing

NF849: Extended Currency Symbol Support
- Extended Currency Symbol Format Options

NF886: Reusable MSO/CICS Control Blocks

NF898: SQL Support for Date-Time Formats
- Date-Time Support for the Relational Data Adapters
- AUTODB2 Support for DDF
- AUTODB2/SQL Support for TIME and TIMESTAMP
  - AUTODB2/SQL Screens That Support DDF and Date-Time Formats
  - Executing AUTODB2 in Batch

SQL Translator Support for Date-Time Fields
Operations on Date-Time Data Supported by the SQL Translator
Extracting Date-Time Components Using the SQL Translator

NF914: LMF Site Code Warning Message

7.1 New Features

NF692: Aggregating and Sorting Report Columns

NF696: Calling Subroutines Written in REXX
Using REXX Subroutines
Compiling FUSREXX Macros in CMS

NF731: Reporting From Independent Paths
Retrieving Data From Multiple Paths
MULTIPATH and SET ALL Combinations
Determining if a Segment Is Required

NF749: HOLD FORMAT INTERNAL

NF750: DATASET in a Master File
DATASET Behavior in FOCUS Data Sources
DATASET Behavior in Fixed-Format Sequential Data Sources
DATASET Behavior in VSAM Data Sources

NF751: Date-Time Data Type
Describing Date-Time Values
Date-Time Display Formats
Specifying Date-Time Values
ACTUAL Formats for Date-Time Values
Setting Date-Time Options

Manipulating Date-Time Values
Comparison and Assignment
Date-Time Functions

NF759: Increased Number of Display Fields

NF755: Using FILEDEF for Creating Extract Files

NF761: Comma Suppress Edit Format Option

NF762: Percent Edit Format Option

NF766: DEFINE Functions
Using DEFINE Functions

NF773: Token Delimited Files

NF777: Two-Gigabyte FOCUS Database Support
Enabling Two-Gigabyte Support

NF777: Partitioned FOCUS Data Sources
Partitioning
Intelligent Partitioning
Specifying an Access File in a FOCUS Master File
The FOCUS Access File
FOCUS Access File Attributes
Describing Joined Files

NF778: Dialogue Manager TRUNCATE Function
Using the Dialogue Manager TRUNCATE Function

NF781: Embedding Text Fields in Headings

NF782: Oracle Data Adapter IXSPACE Setting
Specifying Oracle Index Space Parameters

NF785: The Adabas Write Data Adapter for FOCUS
Activating the Adabas Write Data Adapter
Limitations on Options Described for the Adabas Data Adapter
Fields That Cannot be Updated
Checking Adabas Return Codes and FOCUS Error Message Numbers
Adabas Write Examples
Types of Transaction Processing
Descriptor Considerations
Modifying Data
The MATCH Command
The NEXT Command
INCLUDE, UPDATE, and DELETE Processing
Adabas Transaction Control Within MODIFY
Using the Return Code Variable: FOCERROR
Using the Data Adapter SET ERRORRUN Command
Modifying Data sources Without Unique Keys
Referential Integrity
The MODIFY COMBINE Facility
The LOOKUP Function
The FIND Function
Data Adapter Error Messages and Adabas Response Codes
Adabas Write Data Adapter Error Messages
Adabas Response Codes
Common User Errors

7.0.9 New Features

NF575: Fusion

NF716: Euro Currency Support
- Converting Currencies
- Preparing FOCUS to Process Currency Conversions
  - Creating the Currency Database
  - Identifying Fields That Contain Currency Data
- Activating the Currency Database
- Processing Currency Data

NF744: HOLD FORMAT EXCEL

NF730: Hold Format PDF
- Required Software Configuration
  - Downloading PDF Output

NF654: HOLD From External Sort
- Conditions for Using External Sort to Create a HOLD File

NF597: Aggregation by External Sort
- Conditions for Aggregating with an External Sort

NF728: Changing Retrieval Order with Aggregation

NF655: FOCPROF - The System Wide Profile
- FOCUS Profiles

NF660: Multi-volume Support in MVS FOCUS
- Advantages of Multi-volume Data Sources
- Allocating Multi-volume Data Sources
- Choosing Default Sizes for FOCUS-created Files

NF584: Dynamically Setting the IDMS DBNAME and DICTNAME

NF673: Model 204 Interface Account Split

NF720: SQLJOIN OUTER Setting for Relational Interfaces

NF652: Teradata Interface Kanji Support

NF722: FOCUS Client DNS Names Support

NF656: Controlling REBUILD Messages

NF670: DYNAM Support for Unit Count
Advantages of Multi-volume Data Sources

NF691: Escape Character for LIKE Predicate
Escape Character Capabilities

NF718: DYNAM Support for Existing Relative GDG Numbers
Using DYNAM With Relative GDG Numbers

NF735: Enhancement to ? SET

NF740: Changes to the REBUILD Prompt

NF745: ? PTF Enhancements

NF746: Leading Zeros

NF748: HOLD FORMAT WP With Carriage Control

Index

7.3 New Features

Reporting Enhancements

NF830: Minus Edit Format Option
NF888: Excel 2000
NF926: Controlling Report Page Numbering
NF928: Compiled DEFINEs
NF929: Multivariate Linear Regression
NF930: Holding Missing Values
NF931: Multiple FOLD-LINE
NF932: Including Data in Multiple FOR Rows in FML
NF936: Summary Prefix Operators
NF938: Decimal Alignment of Headings
NF940: SET HNODATA
NF945: Creating Compound PDF or PostScript Reports
NF951: Defining and Displaying FML Hierarchies
NF952: TABLASTPAGE
NF958: StyleSheet Enhancements
NF964: Extensions to FORECAST
NF968: FMLINFO Function
NF971: SET NULL=ON
NF972: NEWPAGE
NF973: Cascading Style Sheets
NF977: IN-RANGES-OF

Reporting Enhancements (continued)

NF979: SET HOLDFORMAT
NF980: SET BLANKINDENT
NF982: SET BYDISPLAY
NF983: Indenting FML Reports
NF989: Excel 97

General Enhancements

NF771: Long Dialogue Manager Variables
NF920: NORMSDST and NORMSINV Subroutines
NF922: AnV Format
NF939: SET SAVEDMASTERS
NF946: &FOCUSER
NF956: Wide Lines
NF961: USERFCHK Setting
NF970: SUFFIX = TAB
NF984: IEDIT Command
NF985: DYNAM Support for New GDG Iterations

Raised Limits

NF923: COMBINE Command Enhancement
NF943: Increased Number of ACROSS Columns
NF944: Increased Space for Column Titles
NF955: Long Qualified Field Names
NF965: Raised Limit for EXTERNAL INDEX
NF981: Increased Number of Sort Headings and Footings
NF995: Raised Limit for IF-THEN-ELSE Conditions

Database Enhancements

NF916: XFOCUS Data Source
NF917: Long Segment Names
NF935: Long Index Names
NF953: File Extensions For Maintain Procedures
NF954: Enhanced Screening Conditions for Maintain
NF959: DATASET for a Segment in a Master File
NF986: GROUP Fields in FOCUS Data Sources
NF987: MDI for FOCUS and XFOCUS Data Sources
NF994: FOCUS SETS From Maintain

Data Adapter Enhancements

- NF861: Increased Row Size Limit for Relational Data Adapters
- NF866: DB2 V7.1 FETCH FIRST n ROWS Support
- NF937: Optimizing Selection of Relational Variable Length Character Data Types
- NF957: FIELDTYPE=R

NF771: Long Dialogue Manager Variables

The lengths of values stored in Dialogue Manager (amper) variables have been increased in the following contexts:

- -READ, -TYPE, and -WRITE commands can operate on values up to 32K bytes long.
- All other Dialogue Manager commands (those that start with a dash), as well as the EX command, and the SQL command can operate on values of up to 4K bytes.

Prior to this feature, the maximum length of a variable value was 256 bytes.

Using Long Strings With -SET

To set the value of a variable with -SET, you need to specify a character string on the right side of the -SET command. The character string that you specify cannot span multiple lines. Therefore, to set a variable to a long string (up to 4K) you may need to specify the long string as the concatenation of shorter strings or variables. This type of expression can span lines.

Example Using -SET to Initialize a Variable to a Long String

The following procedure uses TED to create a variable named &LONG that contains a long string. (Note that with IEDIT, lines can be wider. See *NF984: IEDIT Command* for information):

```
-SET &LONG = 'THIS IS A LONG AMPER VARIABLE. NOTE THAT IN ORDER '
-  |'TO SET ITS VALUE USING -SET, YOU MUST CONCATENATE SHORTER STRINGS, '
-  |'EACH OF WHICH MUST FIT ON ONE LINE.';
-TYPE &LONG
END
```

The output is:

```
 THIS IS A LONG AMPER VARIABLE. NOTE THAT IN ORDER TO SET ITS VALUE
USING -SET,
YOU MUST CONCATENATE SHORTER STRINGS, EACH OF WHICH MUST FIT ON ONE
LINE.
```

Example Using a Long Amper Variable With the EX Command

Consider a FOCEXEC named MYFEX, which references the Dialogue Manager variables &FIELD1, &FIELD2, &FIELD3, &CATEGORY, and &RATING:

```
TABLE FILE MOVIES
PRINT &FIELD1 &FIELD2 &FIELD3
BY CATEGORY BY RATING
WHERE CATEGORY EQ '&CATEGORY'
WHERE RATING EQ '&RATING'
END
```

The following FOCEXEC creates a long variable that contains the command to execute this procedure and supplies values for all of the variables:

```
-SET &LONG = 'EX MYFEX '
-            | 'CATEGORY=ACTION, RATING=R, FIELD1=TITLE/A20, '
-            | 'FIELD2=DIRECTOR, FIELD3=LISTPR ';
-TYPE Request string is: &LONG
&LONG
-RUN
```

The output is:

```
Request string is: EX MYFEX CATEGORY=ACTION, RATING=R, FIELD1=TITLE/A20,
FIELD2=
DIRECTOR, FIELD3=LISTPR

CATEGORY  RATING  TITLE                 DIRECTOR           LISTPR
--------  ------  -----                 --------           ------
ACTION    R       ROBOCOP               VERHOVEN P.         19.98
                  TOTAL RECALL          VERHOVEN P.         19.99
                  RAMBO III             MCDONALD P.         19.95
```

Example Using a Long Amper Variable in a TABLE Command

```
-SET &LONG = 'PRINT TITLE/A15 ' |
-            'WHOLESALEPR AS  PRICE,WHOLE '|
-            'LISTPR AS  PRICE,LIST  ' |
-            'DIRECTOR/A10 RELDATE COPIES '|
-            'BY MOVIECODE AS CODE BY RATING '|
-            'WHERE RATING EQ ''G'' OR ''PG''';
TABLE FILE MOVIES
&LONG
END
```

The output is:

```
                                    PRICE   PRICE
CODE    RATING  TITLE               WHOLE   LIST   DIRECTOR    RELDATE   COPIES
----    ------  -----               -----   -----  --------    -------   ------
001MCA  PG      JAWS                10.99   19.95  SPIELBERG   78/05/13       2
035CBS  PG      CABARET             13.99   19.98  FOSSE B.    73/07/14       1
040ORI  G       BABETTE'S FEAST     13.25   19.98  AXEL G.     88/03/23       1
043DIS  G       SHAGGY DOG, THE     29.99   44.95  BARTON C.   59/01/09       2
081MCA  PG      REAR WINDOW          9.00   19.98  HITCHCOCK   54/12/15       2
082MCA  PG      VERTIGO              9.00   19.98  HITCHCOCK   58/11/25       2
196MCA  PG      BACK TO THE FUT     12.55   19.95  ZEMECKIS R  86/11/19       4
284MGM  G       GONE WITH THE W     40.99   89.95  FLEMING V.  39/06/04       3
305PAR  PG      AIRPLANE             7.85   14.95  ABRAHAMS J  82/09/28       2
306DIS  G       ALICE IN WONDER     12.50   29.95  GEROMINI    51/07/21       2
518MGM  PG      ANNIE HALL          12.55   19.95  ALLEN W.    78/04/16       2
524MGM  G       FIDDLER ON THE      15.00   29.95  JEWISON N.  72/11/16       1
658CBS  PG      BIG                 13.75   19.98  MARSHALL P  90/01/05       4
692PAR  PG      TOP GUN              9.99   14.95  SCOTT T.    88/01/24       2
710VES  PG      FAMILY, THE         32.00   59.95  SCOLA E.    88/06/11       1
732DIS  G       BAMBI               12.00   26.99  DISNEY W.   42/07/03       3
803WAR  PG      DEATH IN VENICE     30.00   59.95  VISCONTI L  73/07/27       1
```

Reference Usage Notes for Long Dialogue Manager Variables

- Long Dialogue Manager variables are not supported in COMPUTE commands, headings, or other portions of non-Dialogue Manager commands.
- The combined length of all local and global variables for each FOCEXEC can be up to 32K.
- Long Dialogue Manager variables cannot be used to extend CRTFORM or -CRTFORM lines beyond the 78-character display limit.

NF830: Minus Edit Format Option

The minus edit format option enables you to display a minus sign to the right of negative numeric data. The option can be defined in a Master File or in a COMPUTE or DEFINE command.

Syntax How to Display Minus Signs

Format	Data	Display
I2-	-21	21-
D7-	-6148	6148-
F7.2-	-8878	8878.00-

Example Describing the Minus Edit Format Option

The following illustrates how to assign a minus sign to a field in a Master File:

```
FIELDNAME=MEMBER_DISCOUNT  ,ALIAS=MDISC  ,FORMAT=D7.2-  ,$
```

The following example illustrates how to assign a minus sign in a COMPUTE command:

```
DEFINE FILE MOVIES
 MDISC/D7.2 = 7.5;
END
TABLE FILE MOVIES
 SUM COMPUTE GROSS_PROFIT = LISTPR - WHOLESALEPR;
             NET_PROFIT/D12.2- = GROSS_PROFIT - MDISC;
 BY  CATEGORY
 PRINT  MDISC AS 'MEMBER,DISCOUNT'
        WHOLESALEPR AS 'WHOLESALE,PRICE'
        LISTPR AS 'LIST,PRICE'
 IF CATEGORY EQ 'ACTION' OR 'MUSICALS' OR 'DRAMA' OR 'SCI/FI';
 IF RECORDLIMIT EQ 10
END
```

In the following output the results of the minus edit option can be seen in the NET_PROFIT column:

```
MEMBER    WHOLESALE   LIST
 CATEGORY    GROSS_PROFIT       NET_PROFIT  DISCOUNT  PRICE       PRICE
 --------    ------------       ----------  --------  ---------   -----
 ACTION              8.96            1.46       7.50      10.99   19.95
 DRAMA               9.98            2.48       7.50      10.00   19.98
 MUSICALS           16.97            5.53-      7.50      13.99   19.98
                                               7.50      13.99   19.98
                                               7.50       9.99   14.98
 SCI/FI             35.32            2.18-      7.50      14.99   19.98
                                               7.50      14.55   19.98
                                               7.50      13.99   19.95
                                               7.50      16.00   24.95
                                               7.50      19.99   29.98
```

Reference Usage Notes for Minus Display Option

- This feature does not perform calculations intended to derive a negative number in the data. It is only a format option that allows numeric data to be displayed with a minus sign.
- A blank space is displayed to the right side of the data when the data is positive in value.

Reference Error Messages for Minus Display Option

Using the minus edit format option with format options B, E, R, T, DMY, MDY, and YMD results in the following error message being issued:

```
(FOC253)     INVALID FORMAT SPECIFICATION ON LEFT HAND SIDE: D12.2R-
```

Using the minus edit format option with two minus signs also results in the preceding error message being displayed.

Placing a minus sign on the right side of data fields in a CRTFORM results in the following error message being issued:

```
(FOC210)     THE DATA VALUE HAS A FORMAT ERROR:
```

An alphabetic character has been found where all numerical digits are required.

NF861: Increased Row Size Limit for Relational Data Adapters

Columns described with ACTUAL=TX in a Master File on OS/390 and z/OS now support text strings up to 32K, for example DB2 CLOB and LONG VARCHAR columns. These columns can be used for read-only operations.

Syntax How to Describe a Text Field in the Master File

```
FIELDNAME=fname, ALIAS=sqlcolname, USAGE=TXnn, ACTUAL=TX ,$
```

where:

fname

Is any valid field name.

sqlcolname

Is the RDBMS column name.

nn

Is the length of an output line for display. The maximum line length is 254 characters.

Example Describing a DB2 CLOB Column

In the Master File:

```
FIELD=CLOB_COL, ALIAS=CLOB_CO, USAGE=TX100, ACTUAL=TX, MISSING=ON,$
```

A request that retrieves this column can display text output up to 32K in length.

NF866: DB2 V7.1 FETCH FIRST n ROWS Support

READLIMIT tests now restrict the size of the answer set returned by the RDBMS.

In prior releases, DB2 constructed a full answer set even if a request included a READLIMIT test. The data adapter then retrieved from the answer set only the number of rows specified in the request.

When accessing DB2 version 7.1 or higher, the data adapter translates a READLIMIT phrase to the appropriate clause to inform the RDBMS that only *n* rows need to be included in the answer set. For $n = 1$, the SQL request contains the clause FETCH FIRST 1 ROW ONLY. The reduction in answer set size reduces response time and enhances performance.

RECORDLIMIT tests do not restrict the size of the answer set returned by DB2. The full answer set is constructed and the data adapter retrieves the number of rows specified by the RECORDLIMIT test.

Note: This feature is not available in DB2 for VM or VSE.

Example Reducing Answer Set Size

The following request turns on the STMTRACE trace component and issues a retrieval request with a READLIMIT phrase. The SQL request contains the FETCH FIRST 2 ROWS clause:

```
SET TRACEUSER=ON
SET TRACEOFF=ALL
SET TRACEON=STMTRACE//CLIENT
TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME CURRENT_SALARY
BY EMP_ID
IF READLIMIT EQ 2
END
```

The generated SQL request contains the FETCH FIRST 2 ROWS clause:

```
 SELECT T1."EID",T1."LN",T1."FN",T1."CSAL" FROM
USER1."EMPINFO" T1 ORDER BY T1."EID" FETCH FIRST 2 ROWS ONLY;
```

If the test specifies READLIMIT EQ 1, the SQL request contains the clause FETCH FIRST 1 ROW ONLY:

```
      SELECT T1."EID",T1."LN",T1."FN",T1."CSAL" FROM
 USER1INFO" T1 ORDER BY T1."EID" FETCH FIRST 1 ROW ONLY;
```

NF888: Excel 2000

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.

Creating Styled Excel 2000 Reports

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 that allows users to create differing views of their data by reiteratively dragging and dropping data fields within their “PivotTable” to employ varied sorts across rows or columns.

Syntax How to Create a Styled Excel 2000 Report

```
[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 filetype 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*.

`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*.

National Language Support with EXL2K

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: 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.

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	19,133,395.00
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	11,273,595.00
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).

Displaying Formatted Dates and Numeric Values

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 displays in a spreadsheet when using 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 the spreadsheet.

```
SET PAGE-NUM=OFF
DEFINE FILE EMPLOYEE
YRHIRED/YY = HIRE_DATE;
MHIRED/MtYY = HIRE_DATE;
TOTSVC/I4C = 2002 - YRHIRED;
END
TABLE FILE EMPLOYEE
"Employee Service Report for 2002"
"Excel 2000 Spreadsheet"
" "
PRINT FIRST_NAME AS 'First Name'
MHIRED 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 Unsupported 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.

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

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`

Designates a particular column to apply wrapping behavior to. If COLUMN is not included in the declaration, wrapping will be applied to the entire report.

`value`

Is one of the following:

`ON`

Turns on data wrapping. This 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 that the column width can be set to. The value represents the measure specified with the UNITS parameter (the default is inches). This setting implies ON. However, the column width is set to the specified width unless the data is wider than the column width, in which case wrapping will occur as for ON.

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=5” 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	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOOD
NORTH	10/22/60	156,931,554.000	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOOD
SOUTH	10/22/60	184,491,442.000	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOOD
WEST	10/22/60	71,649,432.000	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LINE_COST	MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOOD

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 LI	233,492,476.00
MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LI	156,931,554.00
MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LI	184,491,442.00
MULTIPLY QUANTITY TIMES LINE_COST TO CALCULATE LINE_COST_OF_GOODS	MULTIPLY QUANTITY TIMES LI	71,649,432.00

Using the Excel 2000 Formula Option

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, RECAP, and COMPUTE, and for calculations performed by functions.

EXL2K FORMULA is *not* supported with PivotTables (EXL2K PIVOT) 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) displays 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:

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 Name:	Line Total	Quantity:	Line Total	Quantity:	Line Total	Quantity:
6	eMart	$48,833,656.98	196,084	$114,336,646.64	457,064	$163,170,303.62	653,148
7	Audio Expert	$136,936,955.91	531,280	$8,549,478.68	32,566	$145,486,434.59	563,846
8	AV VideoTown	$27,578,810.97	109,504	$17,113,451.48	70,330	$44,692,262.45	179,834
9	City Video	$5,911,253.15	24,565	$9,387,798.14	36,704	$15,299,051.29	61,269
10	Consumer Merchandise	$20,298,468.86	80,701	$5,181,690.34	20,203	$25,480,159.20	100,904
11	TV City	$22,034,204.35	89,148	$23,791,473.06	90,332	$45,825,677.41	179,480
12	Web Sales	$4,027,582.48	12,225			$4,027,582.48	12,225
13	TOTAL	$265,620,932.70	1,043,507	$178,360,538.34	707,199	$443,981,471.04	1,750,706
14							

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.

D4 = =$C4 *$B4

	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	$2,131,786,367,481,040.00

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 000000000) 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, OVRLAY, POSIT, RDUNIF, RANDOM, RJUST, SQRT, SUBSTR, TODAY, TODAYI, UPCASE, YM.
- If you use the COMPUTE command with an unsupported function, an error message is displayed.
- 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.
- 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 and 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

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 display simultaneously as a column and a row field.
- Styling is based on the initial state of your report and retained only on a limited basis when you pivot the output.
- NODATA fields display 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:

B24 = 3364

	A	B	C	D	E	F	G	H	I
1	CENTINV File PivotTable								
2	Sum of Price by Product Across Category								
3	Product Name:	(All)							
4									
5	Sum of Price:	Product Category:							
6	Product Number:	Camcorders	Cameras	CD Players					
7	1004								
8	1006								
9	1008					199			
10	1010	999							
11	1012	899							
12	1014		249						
13	1016		279						
14	1018	399							
15	1020	319							
16	1022	399							
17	1024	349							
18	1026		129						
19	1028		109						
20	1030			169					
21	1032				89				
22	1034						499		
23	1036						299		
24	Grand Total	3,364.00	766.00	169.00	89.00	598.00	798.00	179.00	

PivotTable

PivotTable

Product... Price: Product... Our Cos... Quantit...

Product...

Reference How TABLE Elements Display 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.

Page field item
PivotTable Toolbar-
Contains the
Pivot Cache
Page field
(PAGEFIELDS)
CAR File PivotTable
Sum of Sales by Car Across Model
Data field
(PRINT)
Row field
(BY)
Grand Total
Column field
(ACROSS)
Available fields that
can be dragged into
the PivotTable.
(CACHEFIELDS)

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.
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 displays 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

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:

	A	B	C
1	PivotTable with CACHEFIELDS		
2	Sum of Price by Product Across Category		
3	Sum of Price:		
4	Product Category:	Product Type:	Total
5	Camcorders	Analog	1,466.00
6		Digital	1,898.00
7	Camcorders Total		3,364.00
8	Cameras	Analog	238.00
9		Digital	528.00
10	Cameras Total		766.00
11	CD Players	Digital	169.00
12	CD Players Total		169.00
13	Digital Tape Recorders	Digital	89.00
14	Digital Tape Recorders Total		89.00
15	DVD	Digital	598.00
16	DVD Total		598.00
17	PDA Devices	Digital	798.00
18	PDA Devices Total		798.00
19	VCRs	Analog	179.00
20	VCRs Total		179.00
21	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 that a field specified as a PAGEFIELD cannot be designated anywhere else in the request.

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:

Transferring Excel 2000 Formatted Files Using FTP

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 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 when 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$.XML” PivotTable file.
- Before the FTP operation on MVS, issue a DYNAM FREE for both the “asname” and the “asname$” ddnames.

NF916: XFOCUS Data Source

Note: As FOCUS 7.3 goes production, this feature is still in the process of undergoing the extensive testing effort needed before certifying it. It is, therefore, deactivated in the initial release of FOCUS 7.3. It will be activated in a future service pack after it has completed the testing cycle.

The XFOCUS database is a new database structure that parallels the original FOCUS database, but extends beyond its capabilities with several significant performance and data volume improvements:

- Allows over four times the amount of data per segment instance.
- Holds eight times as much data in a single physical file - up to 16 gigabytes. With partitioning, one logical XFOCUS database may be as large as four terabytes
- With the new multi-dimensional index (MDI) paradigm, sophisticated indexed searches are now possible, with retrieval time improved by as much as 90%. (Performance without the MDI may show only marginal improvement.)

The XFOCUS database has 16K database pages. The SUFFIX in the Master File is XFOCUS. FOCUS databases (SUFFIX=FOC) have 4K database pages.

All existing commands that act on FOCUS files work on XFOCUS files. (See the Usage Notes for some limitations.) No new syntax is required, except for a new SET command and MDI options.

You can convert to an XFOCUS database from a FOCUS database by changing the SUFFIX in the Master File, issuing a SET command to enable this type of data source, and applying the REBUILD utility.

Syntax How to Enable Use of an XFOCUS Data Source

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

where:

ON

Enables creation and use of XFOCUS data sources. This setting does not inhibit or affect the use of FOCUS data sources, it adds the ability to create and use XFOCUS data sources.

OFF

Disables creation and use of XFOCUS data sources. OFF is the default value.

Note: This setting is not supported in an ON TABLE phrase.

Syntax How to Specify an XFOCUS Data Source

```
FILE=name, SUFFIX=XFOCUS, $
```

where:

name

Can be any valid file name.

Syntax How to Control the Number of Pages Used for XFOCUS Data Source Buffers

FOCUS data sources use buffer pages that are allocated by the BINS setting. Buffer pages for XFOCUS data sources are allocated by the XFOCUSBINS setting:

```
SET XFOCUSBINS = n
```

where:

n

Is the number of pages used for XFOCUS data source buffers. Valid values are16 to 1023. The default value is 16.

The memory is not actually allocated until an XFOCUS data source is used in the session. Therefore, if you issue the ? SET XFOCUSBINS query command, you will see the number of pages set for XFOCUS buffers and an indication of whether the memory has actually been allocated (passive for no, active for yes).

ProcedureHow to Create an XFOCUS Data Source

There are two methods for creating an XFOCUS data source. You can either issue the CREATE FILE command or use the HOLD FORMAT XFOCUS command.

Do the following to create an XFOCUS data source using the CREATE FILE command:

1. Create a Master File that specifies SUFFIX=XFOCUS.
2. Issue the CREATE FILE command

   ```
   CREATE FILE name
   ```

   where:

   name

   Is the name of the Master File that specifies SUFFIX=XFOCUS.

Accessing an XFOCUS Data Source

All FOCUS syntax is supported with the following notes:

- An attempt to access a SUFFIX=XFOCUS data source with an earlier release causes an error because SUFFIX=XFOCUS is not recognized as a valid value.
- JOIN commands among suffix FOC, FILETYPE FUSION, and suffix XFOCUS data sources are supported. Master File cross-references are supported to other suffix XFOCUS data sources only.
- The COMBINE command supports SUFFIX=XFOCUS data sources. You can COMBINE SUFFIX FOC and XFOCUS in a single COMBINE.

Reference Usage Notes for the XFOCUS Data Source

- On mainframe platforms, the LRECL and BLKSIZE are both 16384 (16K).
- The file type on VM is XFOCUS.
- Alphanumeric fields with the format A4096 are supported. They are not limited as in SUFFIX=FOC.
- The CALCFILE utility automatically adjusts the calculation algorithm to the suffix type.
- The USE command supports 250 files of mixed XFOCUS and FOC suffixes as long as each type of data source has its own Master File with the correct suffix (FOC for the FOCUS data sources, XFOCUS for the XFOCUS data sources).

  Specify USE...AS for the data sources with suffix FOC and another AS for the data sources with suffix XFOCUS in the same USE.

NF917: Long Segment Names

The length of a segment name in an XFOCUS Master File can be up to 64 characters (see *NF916: XFOCUS Data Source*).

Segment names for non-FOCUS data sources can also be up to 64 characters. However, shorter names must be used if the DBMS does not support 64-character names.

Procedure How to Create an XFOCUS Data Source With Long Segment Names

1. Create a Master File with segment names up to 64 characters long. This Master File must have the attribute SUFFIX=XFOCUS.
2. Issue the CREATE FILE command

   ```
   CREATE FILE name
   ```

   where:

   name

   Is the name of the Master File that specifies SUFFIX=XFOCUS.

Reference Usage Notes for Long Segment Names

- To change a segment name in an existing XFOCUS data source, you must issue the REBUILD command against the data source.
- If you issue the ? FDT query command, only the first eight characters display for the segment name.
- Segment names used with SU and HLI are limited to eight characters.

- Each XFOCUS data source has a File Definition Table (FDT) that contains an entry for each segment and indexed field. The maximum number of entries allowed for segments and indexed fields for XFOCUS data sources 189 .

  When this limit is exceeded, the following error message is generated:

  ```
  (FOC36347)  UNABLE TO CREATE FOCUS FILE %1: FDT OVERFLOW %2
  ```

  There is a limit to the total number of segments and indices that can exist in a Master. Please check the IBI documentation for exact limits.

  In the message, %1is the name of the Master File and %2 is the number of characters that need to be removed from the total length of the segment and index names.

Example Specifying a Long Segment Name in an XFOCUS Master File

```
FILENAME=TEST1, SUFFIX=XFOCUS,
SEGNAME=EMPLOYEE_INFORMATION_SEGMENT, SEGTYPE=S1,$
FIELDNAME=FIELD1 ,ALIAS=FIELD1A ,FORMAT=A20     ,$
```

NF920: NORMSDST and NORMSINV Subroutines

The NORMSDST subroutine calculates the cumulative standard normal distribution function.

Given a point x_0, NORMSDST returns a value that represents the area to the left of point x_0 under the standard normal distribution curve. This area corresponds to the probability that an arbitrary point, x, is less than or equal to x_0:

The value returned by NORMSDST also represents the percentage of points that are less than or equal to x_0, making this subroutine useful for determining percentiles of normally distributed data.

Given an area or probability on the standard normal curve, the NORMSINV subroutine returns the x value that corresponds to the upper boundary of that area.

NORMSINV is the inverse of NORMSDST. That is, NORMSINV(NORMSDST(x)) = x.

Characteristics of the Normal Distribution

Many common measurements tend to be normally distributed. A plot of normally distributed data values approximates a bell-shaped curve:

The two measures needed to describe any normal distribution are the mean and the standard deviation:

- The mean is the point at the center of the curve.

- The standard deviation describes the spread of the curve. It is the distance from the mean to the point of inflection (point where the curve changes direction).

The *standard* normal distribution is a normal distribution that has a mean of 0 and a standard deviation of 1. The total area under this curve is 1. A point on the x-axis of the standard normal distribution is called a *z-score* or normalized value. Assuming that your data is normally distributed, you can convert a data point (raw score) to a z-score in order to find the percentage of scores that are less than or equal to the raw score.

If you use the z-score as input to the NORMSDST function, the value returned is the percent of scores less than or equal to the z-score (probability that a score is less than or equal to the z-score).

The inverse of the NORMSDST subroutine is called NORMSINV. It takes a percent of the curve and returns the z-score that marks the upper boundary of that portion of the curve.

ProcedureHow to Convert a Value to a Z-Score

You can convert a value (raw score) from your normally distributed data to the equivalent point on the standard normal distribution (z-score) as follows:

```
z = (raw_score - mean)/standard_deviation
```

To convert from a z-score back to a raw score, use the following formula:

```
raw_score = z * standard_deviation + mean
```

where:

`z`

Is the normalized data point, called a z-score.

`raw_score`

Is the raw (actual) data value.

Syntax How to Calculate the Mean of a Set of Values

The mean of data points x_i, where i is from 1 to n is:

$(\Sigma x_i)/n$

Syntax How to Calculate the Standard Deviation of a Set of Values

The standard deviation of data points x_i, where i is from 1 to n is:

$SQRT(((\Sigma x_i^2 - (\Sigma x_i)^2/n)/(n - 1)))$

Syntax How to Calculate the Cumulative Standard Normal Distribution Function

```
NORMSDST(z, 'D8');
```

where:

z

Is a z-score (x value from a standard normal distribution).

The value returned by the subroutine is double precision. You can assign it to a field with any valid numeric format.

Syntax How to Calculate the Inverse Cumulative Standard Normal Distribution Function

```
NORMSINV(p, 'D8');
```

where:

p

Is a number between 0 and +1 (cumulative probability from a standard normal distribution).

The value returned by the subroutine is double precision. You can assign it to a field with any valid numeric format.

Example Using the NORMSDST and NORMSINV Subroutines

This request uses the GGPRODS data source. This data source contains a very small set of points which cannot be assumed to be normally distributed and are used only for demonstration purposes.

The field used is SIZE, which is converted to double precision (as X) for use with the AVE. and ASQ. prefix operators. Because the SIZE values are not normalized, a two-part request is required:

- The SUM command calculates the mean and standard deviation.
- The PRINT command uses the mean and standard deviation to convert the field values to normalized scores for input to NORMSDST.
- The NORMSINV subroutine is used to return the original z-score. The mean and standard deviation are used to convert the z-score back to a SIZE value:

```
DEFINE FILE GGPRODS
-* CREATE AN APPROPRIATE NUMERIC SORT FIELD
PRODNO/I4 = DECODE PRODUCT_ID('B141'  45 'B142'  42 'B144'  44
             'F101' 101 'F102' 102 'F103' 103 'G100' 140
             'G104' 144 'G110' 150  'G121' 151);
-* CONVERT SIZE FIELD TO DOUBLE PRECISION
X/D12.5 = SIZE;
END

TABLE FILE GGPRODS

SUM X NOPRINT CNT.X NOPRINT
-* CALCULATE MEAN AND STANDARD DEVIATION
COMPUTE N0/D12.5       =  CNT.X; NOPRINT
COMPUTE MEAN/D12.5     =  AVE.X; NOPRINT
COMPUTE VARIANCE/D12.5 = ((N0*ASQ.X) - (X*X/N0))/(N0-1); NOPRINT
COMPUTE STDEV/D12.5    = SQRT(VARIANCE); NOPRINT

PRINT SIZE X NOPRINT
-* COMPUTE Z-SCORES AND USE AS INPUT TO NORMSDST FUNCTION
-* THEN USE RETURNED VALUES AS INPUT TO NORMSINV FUNCTION
-* AND CONVERT BACK TO DATA VALUES
COMPUTE Z/D12.5      = (X - MEAN)/STDEV;
COMPUTE NORMSD/D12.5 = NORMSDST(Z, 'D8');
COMPUTE NORMSI/D12.5 = NORMSINV(NORMSD, 'D8');
COMPUTE DSIZE/D12    = NORMSI * STDEV + MEAN;
BY PRODNO NOPRINT
END
```

The output is:

```
Size              Z         NORMSD         NORMSI            DSIZE
----              -         ------         ------            -----
  12        -.80273         .21106        -.80273               12
  12        -.80273         .21106        -.80273               12
  16        -.07298         .47091        -.07298               16
  20         .65678         .74434         .65678               20
  24        1.38654         .91721        1.38654               24
  20         .65678         .74434         .65678               20
  24        1.38654         .91721        1.38654               24
  16        -.07298         .47091        -.07298               16
  12        -.80273         .21106        -.80273               12
   8       -1.53249         .06270       -1.53249                8
```

Reference Usage Notes for the NORMSDST and NORMSINV Subroutines

- Data must come from a normal distribution.
- Results are accurate to six significant digits.
- NORMSINV uses an iterative technique. If it does not converge after 100 iterations, the function returns a zero value.
- Return values are generated as double precision. They can be assigned to F, P, or D formatted fields only.

NF922: AnV Format

The character format A*n*V is supported in Master Files for FOCUS, XFOCUS, Fusion, and relational data sources. This format is used to represent the VARCHAR (variable length character) data types supported by relational database management systems.

For relational data sources, A*n*V keeps track of the actual length of a VARCHAR column. This information is especially valuable when the value is used to populate a VARCHAR column in a different RDBMS. It affects whether trailing blanks are retained in string concatenation and, for Oracle, string comparisons (the other relational engines ignore trailing blanks in string comparisons).

In a FOCUS, Fusion, or XFOCUS data source, A*n*V does not provide true variable length character support. It is a fixed-length character field with an extra two leading bytes to contain the actual length of the data stored in the field. This length is stored as a short integer value occupying two bytes.

Note: Throughout the remainder of this document, the maximal length (*n*) of an A*n* or A*n*V field will be referred to as its *size* or its *actual size*. The actual length of an A*n*V field (the value stored in the first two bytes) will be referred to as its *length*. For an A*n* field, length is equal to size.

Syntax How to Specify AnV Fields in a Master File

```
FIELD=name, ALIAS=alias, USAGE=AnV [,ACTUAL=AnV] , $
```

where:

n

Is the size (maximum length) of the field. It can be from 1 to 4093. Note that because of the additional two bytes used to store the length, an A4093V field is actually 4095 bytes long. A size of zero (A0V) is not supported. The length of an instance of the field can be zero.

Note: HOLD FORMAT ALPHA creates an ACTUAL format of A*n*W in the Master File. For information, see *Propagating an AnV Field to a HOLD File*.

Example Specifying AnV Format in a Master File

The following represents a VARCHAR field in a Master File for a DB2 data source with size 200:

```
$ VARCHAR FIELD USING AnV
 FIELD=VARCHAR200, ALIAS=VC200, USAGE=A200V, ACTUAL=A200V, MISSING=ON ,$
```

The following represents an A*n*V field in a Master File for a FOCUS data source with size 200:

```
FIELD=ALPHAV, ALIAS=AV200, USAGE=A200V, MISSING=ON ,$
```

If a data source has an A*n*V field, specify the following in order to create a HOLD FORMAT ALPHA file without the length designator:

```
FIELD=ALPHA, USAGE=A25, ACTUAL=A25V, $
```

or

```
DEFINE …
  ALPHA/A25 = VARCHAR ;
END
```

or

```
COMPUTE ALPHA/A25 = VARCHAR ;
```

In order to alter or create a Master File to include A*n*V, the data must be converted and the length added to the beginning of the field. For example, issue a HOLD command when the field is described as follows:

```
FIELD=VARCHAR, ,USAGE=A25V, ACTUAL=A25, $
```

or

```
DEFINE ...
  VARCHAR/A25V = ALPHA ;
END
```

or

```
COMPUTE VARCHAR/A25V = ALPHA ;
```

Reference Usage Notes for AnV Format

- Maintain does not support the A*n*V format. Otherwise, A*n*V can be used anywhere that A*n* can be used, except for the restrictions listed in these notes.
- Full FOCUS and SQL operations are supported with this data type, including CREATE FILE for relational data sources.
- Joins are not supported between A*n* and A*n*V fields.
- DBCS characters are supported. As with the A*n* format, the number of characters must fit within the 4K data area.
- COMPUTE and DEFINE generate the data type specified on the left-hand side.
- Conversion between A*n*V and TX fields is not supported.
- StyleSheets do not support A*n*V fields.

- A*n*V fields cannot have date display options.

Propagating an AnV Field to a HOLD File

When a user propagates an A*n*V field to a sequential data source using the HOLD FORMAT ALPHA command, the two-byte integer length is converted to a six-digit alphanumeric length. The field in the HOLD file consists of this six-digit number followed by the character data. The format attributes for this field are:

```
... USAGE=AnV, ACTUAL=AnW
```

A*n*W is created as a by-product of HOLD FORMAT ALPHA; however it can be read and used for input as necessary. The number of bytes occupied by this field in the HOLD file is 6+*n*.

In a binary HOLD file, the USAGE and ACTUAL formats are A*n*V, although the ACTUAL format may be padded to a full 4-byte word. The number of bytes occupied by this field in the HOLD file is 2+*n*.

When an A*n*V field is input into a data source, all bytes in the input field beyond the given length are ignored; these bytes are set to blanks as part of the input process.

When a user creates a relational data source using the HOLD FORMAT *sqlengine* command, the A*n*V field generates a VARCHAR column in the relational data source.

Example Propagating an AnV Field to a FORMAT ALPHA HOLD File

The following request creates a field named TITLEV with A39V format and propagates it to a FORMAT ALPHA HOLD file. Note that TITLEV is not a true variable length field, but is used to illustrate how the A*n*V format is propagated to a HOLD file:

```
TABLE FILE MOVIES
PRINT DIRECTOR
COMPUTE TITLEV/A39V = TITLE;
BY CATEGORY
ON TABLE HOLD FORMAT ALPHA
END
```

The generated Master File follows:

```
FILE=HOLD            ,SUFFIX=FIX
SEGNAME=HOLD     ,SEGTYPE=S01
FIELDNAME    =CATEGORY            ,E01          ,A8        ,A08       ,$
FIELDNAME    =DIRECTOR            ,E02          ,A17       ,A17       ,$
FIELDNAME    =TITLEV              ,E03          ,A39V      ,A39W      ,$
```

Example Creating a DB2 Data Source With a VARCHAR Column

```
TABLE FILE MOVIES
PRINT MOVIECODE DIRECTOR
COMPUTE TITLEV/A39V = TITLE;
BY CATEGORY
ON TABLE HOLD AS DB2VAR FORMAT DB2
END
```

The generated CREATE TABLE statement creates a VARCHAR column for TITLEV:

```
 CREATE TABLE DB2VAR( "CLASS" CHAR (8),"FOCLIST" INTEGER  NOT
NULL ,"MCOD" CHAR (6) ,"DIR" CHAR (17),"MTL" CHAR (39),
"TITLEV" VARCHAR (39)  ) ;
 CREATE  UNIQUE INDEX DB2VARIX ON DB2VAR ("CLASS"  ASC,
```

The generated Master File follows:

```
FILE=DB2VAR          ,SUFFIX=SQL
SEGNAME=SEG01   ,SEGTYPE=S0
FIELDNAME   ='CATEGORY'          ,'CLASS'        ,A8        ,A8        ,$
FIELDNAME   =FOCLIST             ,FOCLIST        ,I5        ,I4        ,$
FIELDNAME   ='MOVIECODE'         ,'MCOD'         ,A6        ,A6        ,$
FIELDNAME   ='DIRECTOR'          ,'DIR'          ,A17       ,A17       ,$
FIELDNAME   ='TITLE'             ,'MTL'          ,A39       ,A39       ,$
FIELDNAME   ='TITLEV'            ,'TITLEV'       ,A39V      ,A39V      ,$
```

Using AnV Fields in Expressions

As an alphanumeric type, an A*n*V field can be used in arithmetic and logical expressions in the same way that the A*n* type is used.

- An expression that contains A*n*V type fields can be of either the A*n*V or A*n* type.
- The type that results from the expression depends on the specific type of operation, as described in subsequent sections

Using Concatenation With AnV Fields

If either of the operands in a concatenation between two fields is an A*n*V 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 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 A*n* field in the concatenation, the size and length are equal.

Using the EDIT Function With AnV Fields

The following expression results in an A*n*V format only when *x* has A*n*V 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 A*n*V field to an integer value when *x* has A*n*V format.

Using CONTAINS and OMITS With AnV Fields

The only difference in evaluation of the CONTAINS and OMITS operators with A*n*V 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
`Z CONTAINS Y`	FALSE
`X CONTAINS Z`	TRUE
`Z CONTAINS Z`	TRUE
`Z OMITS Y`	TRUE
`X OMITS Z`	FALSE
`Z OMITS Z`	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 A*n* type fields, operations are evaluated on the assumption that the shorter operand is padded with blanks.

Therefore, even an empty A*n*V field, Z, is compared as a field consisting of all blanks. In the following examples, Z is an empty A*n*V field instance and X is an A*n*V 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
`X GE Z`	TRUE

Using the DECODE Function With AnV Fields

```
DECODE alphafield (value 'result'...
```

The use of either an A*n* or A*n*V field with DECODE causes a result of type A*n* as long as the *result* part of the value-result pairs is provided as a constant. (Constants are type A*n*.)

Using the Assignment Operator With AnV Fields

There are three situations to consider when using the assignment operator with the A*n*V format: A*n*V data type on the right hand side only, A*n*V data type on both sides, and A*n*V 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 A*n* field.
- The size of the A*n*V result does *not* prevent assignment to a shorter A*n* 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 A*n*V evaluation result does *not* prevent assignment to a shorter A*n*V 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 A*n*V field. An error is generated if the result is longer than *n*.

Using AnV Fields With Functions

A*n*V fields can be used as arguments to all Information-Builders-supplied functions that expect alphanumeric arguments. An A*n*V input parameter is treated as an A*n* parameter and is padded with blanks to its declared size (*n*). If the last parameter specifies an A*n*V format, the function result is converted to type A*n*V with actual length set equal to its size.

Note: Many functions require both an alphanumeric string and its length as input arguments. If the supplied string is stored in an A*n*V field, you still must supply a length argument to satisfy the requirements of the function's calling sequence. However, the length that will be used in the function's calculations is the actual length stored as the first two bytes of the A*n*V field.

Six functions are designed to work specifically with the A*n*V data type: LENV, LOCASV, POSITV, SUBSTV, TRIMV, and UPCASV.

In general, any input argument can be a field or a literal. In most cases numeric input arguments to these functions are supplied as literals, and there is no reason not to supply an integer value. However, if the value is not an integer, it is truncated to an integer value regardless of whether it was supplied as a field or a literal.

TRIMV: Removing Characters From a String

TRIMV removes a pattern from a string and is similar to TRIM; however both the input string and the pattern of characters to remove can have A*n*V format.

Note: TRIMV is useful for converting an A*n* field to an A*n*V field (with the length bytes containing the actual length of the data up to the last non-blank character).

Syntax How to Remove Characters From a String

`TRIMV(`*`trim_where, string, slength_limit, pattern, plength_limit, outfield`*`)`

where:

`trim_where`

Alphanumeric

Is one of the following, which indicates where to remove the pattern:

`'L'` removes leading occurrences.

`'T'` removes trailing occurrences.

`'B'` removes both leading and trailing occurrences.

`string`

Alphanumeric.

Is the source field or an alphanumeric constant enclosed in single quotation marks. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field. If *slength_limit* is smaller than the actual length, the source string is truncated to this upper limit.

`slength_limit`

Numeric

Is a positive numeric value. The integer part is an upper limit for the length of the input string.

`pattern`

Alphanumeric

Is a field or an alphanumeric constant enclosed in single quotation marks that is the pattern to remove. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field. If *plength_limit* is smaller than the actual length, the pattern is truncated to this limit.

`plength_limit`

Numeric

Is a positive numeric value. The integer part is an upper limit for the length of the pattern.

outfield

Alphanumeric

Is either a field with A*n*V or A*n* format for the returned substring or the format of the output value enclosed in single quotation marks.

If the format of *outfield* is A*n*V, the actual length is equal to the number of characters left after trimming.

Example Creating an A*n*V Field by Removing Trailing Blanks

The TRIMV function creates an A*n*V field named TITLEV by removing trailing blanks from the TITLE value:

```
TABLE FILE MOVIES
PRINT DIRECTOR
COMPUTE TITLEV/A39V = TRIMV('T', TITLE, 39, ' ', 1, TITLEV);
BY CATEGORY
END
```

LENV: Returning the Length of an Alphanumeric Field

LENV returns the actual length of an A*n*V input field or the size of an A*n* field.

Syntax How to Find the Length of an Alphanumeric Field

LENV(*string, outfield*)

where:

string

Alphanumeric

Is the source field or an alphanumeric constant enclosed in single quotation marks. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field.

outfield

Integer

Field that receives the actual length or the format of the output value enclosed in single quotation marks.

Example Finding the Length of an A*n*V Field

The TRIMV function creates an A*n*V field named TITLEV by removing trailing blanks from the TITLE value. The LENV function returns the actual length of each instance of TITLEV to the ALEN field:

```
TABLE FILE MOVIES
PRINT
COMPUTE TITLEV/A39V = TRIMV('T', TITLE, 39, ' ', 1, TITLEV);
        ALEN/I2 = LENV(TITLEV,ALEN);
BY CATEGORY NORPINT
WHERE CATEGORY EQ 'CHILDREN'
END
```

The output is:

```
TITLEV                                  ALEN
------                                  ----
SMURFS, THE                               11
SHAGGY DOG, THE                           15
SCOOBY-DOO-A DOG IN THE RUFF              28
ALICE IN WONDERLAND                       19
SESAME STREET-BEDTIME STORIES AND SONGS   39
ROMPER ROOM-ASK MISS MOLLY                26
SLEEPING BEAUTY                           15
BAMBI                                      5
```

LOCASV: Creating a Variable Length Lower Case String

LOCASV converts alphabetic characters to lower case and is similar to LOCASE. However, LOCASV can return A*n*V output whose actual length is the lesser of the actual length of the A*n*V input field and an input parameter that specifies the length limit.

Syntax How to Create a Variable Length Lower Case String

`LOCASV(length_limit, string, outfield)`

where:

length_limit

Numeric

Is a positive constant or a field whose integer portion represents the size and, therefore, the upper limit for the length of the input string.

`string`

Alphanumeric

Is the field or alphanumeric constant enclosed in single quotation marks that is to be converted to lower case. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field. If *length_limit* is smaller than the actual length, the source string is truncated to this upper limit.

`outfield`

Alphanumeric

Is either a field with A*n*V or A*n* format for the returned lower case string or the format of the output value enclosed in single quotation marks.

If the format of *outfield* is A*n*V, then the actual length returned is equal to the smaller of the input string length and *length_limit*.

POSITV: Finding the Beginning of a Variable Length Substring

POSITV is similar to POSIT. However, the lengths of its A*n*V parameters are based on the actual lengths of those parameters in comparison with two other parameters that specify their sizes.

Syntax How to Find the Beginning of a Variable Length Substring

`POSITV(parent, in_limit, substring, sub_limit, outfield)`

where:

parent

Alphanumeric

Is the field or alphanumeric constant enclosed in single quotation marks containing the substring whose position you want to find. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field. If *in_limit* is smaller than the actual length, the source string is truncated to this upper limit.

in_limit

Numeric

Is a positive constant or a field whose integer portion represents the size and, therefore, the upper limit for the length of the input string.

substring

Alphanumeric

Is a field name or an alphanumeric constant enclosed in single quotation marks that contains the substring whose position you want to locate. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field. If *sub_limit* is smaller than the actual length, the source string is truncated to this upper limit.

sub_limit

Numeric

Is a positive constant of a field whose integer portion represents the size and, therefore, the upper limit for the length of the substring.

outfield

Integer

Is the name of the field to which the position is returned or the format of the output value enclosed in single quotation marks.

Example Finding the Starting Position of a Variable Length Pattern

In the following request, POSITV finds the starting position of a trailing definite or indefinite article in a movie title (such as ", THE" in SMURFS, THE). First TRIMV removes the trailing blanks from the title so that the article will be the trailing pattern:

```
DEFINE FILE MOVIES
  TITLEV/A39V = TRIMV('T',TITLE, 39,' ', 1, TITLEV);
  PSTART/I4 = POSITV(TITLEV,LENV(TITLEV,'I4'), ',', 1,'I4');
  PLEN/I4 = IF PSTART NE 0 THEN LENV(TITLEV,'I4') - PSTART +1
                    ELSE 0;
END
TABLE FILE MOVIES
  PRINT TITLE
   PSTART AS 'Pattern,Start' IN 25
   PLEN AS 'Pattern,Length'
BY CATEGORY  NOPRINT
WHERE PLEN NE 0
END
```

The output is:

```
                          Pattern  Pattern
TITLE                     Start    Length
-----                     -------  -------
SMURFS, THE                     7        5
SHAGGY DOG, THE                11        5
MALTESE FALCON, THE            15        5
PHILADELPHIA STORY, THE        19        5
TIN DRUM, THE                   9        5
FAMILY, THE                     7        5
CHORUS LINE, A                 12        3
MORNING AFTER, THE             14        5
BIRDS, THE                      6        5
BOY AND HIS DOG, A             16        3
```

SUBSTV: Extracting a Variable Length Substring

SUBSTV extracts a substring from a string and is similar to SUBSTR. However, the end position for the string is calculated from the starting position and the substring length. Therefore, it has fewer parameters than SUBSTR. Also, the actual length of the output field if it is an A*n*V field is determined based on the substring length.

Syntax How to Extract a Variable Length Substring

`SUBSTV(`*`in_limit, parent, start, sublength, outfield`*`)`

where:

`in_limit`

Numeric

Is a positive constant or a field whose integer portion represents the size and, therefore, the upper limit for the length of the input string.

`parent`

Alphanumeric

Is the field or alphanumeric constant enclosed in single quotation marks containing the substring you want to extract. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field. If *in_limit* is smaller than the actual length, the source string is truncated to this size. The final length value determined by this comparison will be referred to as *p_length* (see the description of the *outfield* parameter).

`start`

Numeric

Is a positive constant or a field whose integer portion represents the starting position of the substring. Note that the starting position can exceed the input string length.

`sublength`

Numeric

Is a positive constant or a field whose integer portion represents the length of the substring. The end position of the substring is *end* =*start* + *sublength* -1. Note that the ending position can exceed the input string length depending on the provided values for *start* and *sublength* provided.

`outfield`

Alphanumeric

Is either a field with A*n*V or A*n* format for the returned substring or the format of the output value enclosed in single quotation marks.

If the format of *outfield* is A*n*V, the actual length, *outlen*, is computed as follows from the values for *end*, *start*, and *p_length* (see the *parent* parameter):

If end > p_length or end < start, then outlen = 0 otherwise, outlen = end - start + 1.

Example Extracting a Variable Length Substring

The following request extracts a trailing definite or indefinite article from a movie title (such as ", THE" in "SMURFS, THE"). First it trims the trailing blanks so that the article is the trailing pattern. Next it finds the starting position and length of the pattern. Then SUBSTV extracts the pattern and TRIMV trims the pattern from the title:

```
DEFINE FILE MOVIES
  TITLEV/A39V = TRIMV('T',TITLE, 39,' ', 1, TITLEV);
  PSTART/I4 = POSITV(TITLEV,LENV(TITLEV,'I4'), ',', 1,'I4');
  PLEN/I4 = IF PSTART NE 0 THEN LENV(TITLEV,'I4') - PSTART +1
                   ELSE 0;
  PATTERN/A20V= SUBSTV(39, TITLEV, PSTART, PLEN, PATTERN);
 NEWTIT/A39V=TRIMV('T',TITLEV,39,PATTERN,LENV(PATTERN,'I4'),NEWTIT);
END
TABLE FILE MOVIES
  PRINT TITLE
   PSTART AS 'Pattern,Start' IN 25
   PLEN AS 'Pattern,Length'
  NEWTIT AS 'Trimmed,Title' IN 55
BY CATEGORY  NOPRINT
WHERE PLEN NE 0
END
```

The output is:

```
                                                        Trimmed
TITLE                    PATTERN PSTART  PLEN          Title
-----                    ------- ------  ----          -------
SMURFS, THE              , THE        7     5          SMURFS
SHAGGY DOG, THE          , THE       11     5          SHAGGY DOG
MALTESE FALCON, THE      , THE       15     5          MALTESE FALCON
PHILADELPHIA STORY, THE  , THE       19     5          PHILADELPHIA STORY
TIN DRUM, THE            , THE        9     5          TIN DRUM
FAMILY, THE              , THE        7     5          FAMILY
CHORUS LINE, A           , A         12     3          CHORUS LINE
MORNING AFTER, THE       , THE       14     5          MORNING AFTER
BIRDS, THE               , THE        6     5          BIRDS
BOY AND HIS DOG, A       , A         16     3          BOY AND HIS DOG
```

UPCASV: Creating a Variable Length Upper Case String

UPCASV converts alphabetic characters to upper case like UPCASE. However, UPCASV can return A*n*V output whose actual length is the lesser of the actual length of the A*n*V input field and an input parameter that specifies the size.

Syntax How to Create a Variable Length Upper Case String

`UPCASV(length_limit, string, outfield)`

where:

length_limit

Numeric

Is a positive constant or a field whose integer portion represents the size and, therefore, the upper limit for the length of the input string.

string

Alphanumeric

Is the field or alphanumeric constant enclosed in single quotation marks that is to be converted to upper case. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field. If *length_limit* is smaller than the actual length, the source string is truncated to this size.

outfield

Alphanumeric

Is either a field with A*n*V or A*n* format for the returned upper case string or the format of the output value enclosed in single quotation marks.

If the format of *outfield* is A*n*V, then the actual length returned is equal to the smaller of the input string length and *length_limit*.

Inputting AnV Fields With the MODIFY FIXFORM Command

Two FIXFORM format options support the A*n*V format. The input fields are always fixed length. They may be padded with blanks:

FIXFORM Option	Description
/*n*W	Length of input field is *n*+6 bytes. The first six bytes contain the length of the character data within the subsequent *n* bytes. Use for inputting data from HOLD FORMAT ALPHA files.
/*n*V	Length of input is *n*+2 bytes. The first two bytes are binary and contain the length of the character data within the subsequent *n* bytes. Use for inputting data from binary HOLD files.

If data is input to an A*n*V field using the /*n* FIXFORM option without the V or W option, the length of each instance is determined by removing all trailing blanks.

NF923: COMBINE Command Enhancement

You can now use the COMBINE command to access up to 63 data sources in a single MODIFY request. Each COMBINE structure can include up to 64 segments, the first of which is the virtual root segment created by the COMBINE command.

COMBINE Command Support

The COMBINE command is supported for the following types of data sources:

- FOCUS
- XFOCUS
- Fusion®
- Relational
- Adabas®

The command creates one logical structure from physically separate data sources. The COMBINE structure lasts for the duration of the FOCUS session.

While data sources in a COMBINE structure can have different suffixes, you can only combine a FOCUS, XFOCUS, or Fusion data source to data sources of the same type (although FOCUS and XFOCUS data sources can be referenced in the same COMBINE). The combined structure can include 63 data sources each consisting of a single segment, or a combination of data sources whose segments total to 63 (the addition of the root makes the total 64).

The ? COMBINE query shows up to 63 entries and the CHECK FILE command shows up to 64 segments and files.

Syntax How to Create a COMBINE Structure

```
COMBINE FILES file1 [PREFIX pref1|TAG tag1] [AND]
   .
   .
   .
             filen [PREFIX prefn|TAG tagn] AS asname
```

where:

file1 - filen

Are the Master File names for the data sources you want to modify. You can specify up to 63 names.

pref1 - prefn

Are prefix strings for each data source; up to four characters. They provide uniqueness for field names. You cannot mix TAG and PREFIX in a COMBINE structure.

tag1 - tagn

Are aliases for the Master File names; up to eight characters. FOCUS uses the tag name as the qualifier for fields that refer to that data source in the combined structure. You cannot mix TAG and PREFIX in a COMBINE, and you can only use TAG if FIELDNAME is set to NEW or NOTRUNC.

AND

Is an optional word to enhance readability.

asname

Is the required name of the combined structure to use in MODIFY procedures and CHECK FILE commands.

Once you enter the COMBINE command, you can modify the combined structure.

Reference Usage Notes for COMBINE

All MODIFY code compiled in releases prior to FOCUS 7.3 must be re-compiled.

Reference Error Messages for COMBINE

(FOC214) **MORE THAN 64 SEGMENTS OR 189 SEGMENTS+INDEXES DESCRIBED:**

A Structure can only contain a total of 64 segments or a total sum of 189 segments, indexes, and text fields in any combination.

NF926: Controlling Report Page Numbering

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 on the report. &FOCNEXTPAGE is a system variable whose value is one greater than the last page on the prior report.

Example Setting the Number of the First Page on 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
```

NF928: Compiled DEFINEs

Compiling DEFINE expressions into machine code provides faster processing.The MVS and VM environments have had an expression compiler for several releases. A new DEFINE compiler has been introduced on MVS and VM in FOCUS 7.3.

Therefore, on MVS and VM two expression compilers are available. By issuing the appropriate command you can select one of them or disable expression compilation. Both compilers cannot be active for the same request:

- The new 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 old compiler. It provides no compilation for COMPUTE expressions in TABLE or MODIFY procedures. Compilation takes place at TABLE runtime. This compiler is invoked by issuing the command SET DEFINES = COMPILED.
- The compiler available in prior releases is invoked with the command SET COMPUTE = NEW (the default). This compiler provides expression compilation for DEFINE, IF, WHERE, and COMPUTE commands in both TABLE and MODIFY 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 new 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.

Once the new 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 the 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.

Syntax How to Compile DEFINE Expressions

Issue the following command in FOCPARM, a FOCEXEC, or at the command line

```
SET DEFINES = {COMPILED|OLD}
```

where:

`COMPILED`

Implements expression compilation at request runtime, compiling only those DEFINEs that are used in the request.

`OLD`

Leaves expression compilation up to the control of the current SET COMPUTE value. OLD is the default 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
```

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.
- Issuing the SET DEFINES=OLD command deactivates the new compiler and automatically activates the old compiler.
- 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.

The old compiler is recommended for MODIFY requests and those TABLE requests that use COMPUTE rather than DEFINE fields.

If a TABLE request uses both COMPUTE and DEFINE fields, it is almost certain to execute faster using the new compiler, especially if it retrieves a large number of records. Also, 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:
  - Exponentiation.
  - Functions. However, expressions that use the following functions can be compiled: YMD, DMY, INT, and DECODE.
  - MISSING, CONTAINS, OMITS, 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 will be handled without compilation. No message will be generated indicating that compilation did not take place. To determine whether compilation did take place, issue the ? COMPILE command.

NF929: Multivariate Linear Regression

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 (x1, x2, x3) are related by the following linear equation:

```
y = a1*x1 [+ a2*x2 [+ a3*x3]] + 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 (a1, a2, a3, 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|TABLE} 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. Format I (Integer) is not supported.

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 sort field used for REGRESS must be a numeric or smart date field. 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.
- The SUM command is not supported with REGRESS.
- 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.

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
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
END

TABLE FILE GGSALES
PRINT BUDUNITS UNITS BUDDOLLARS DOLLARS
ON TABLE RECAP ESTDOLLARS/F8 = REGRESS(3, BUDUNITS,
                               UNITS, BUDDOLLARS, DOLLARS);
WHERE SYEAR EQ 97
WHERE CATEGORY EQ 'Coffee'
WHERE REGION EQ 'West'
WHERE UNITS GT 1600 AND UNITS LT 1700
END
```

The output is:

Budget Units	Unit Sales	Budget Dollars	Dollar Sales	ESTDOLLARS
1796	1696	17960	25440	23381
1463	1663	21945	19966	18919
1759	1615	17590	17765	20850
1403	1601	21045	17611	16530
1825	1695	23725	25425	24316
1613	1685	22582	18535	21365
1568	1682	23520	25230	20841
1733	1647	17330	24705	21313
1870	1620	24310	24300	23033
1665	1678	21645	23492	21722
1725	1669	22425	21697	22307
1457	1671	21855	20052	19038
1662	1674	24930	18414	21931
1556	1643	18672	18073	19201
1653	1694	21489	16940	21963
1473	1616	14730	16160	17095

Reference REGRESS Error Messages

(FOC36323) EXPRESSIONS ARE NOT PERMITTED IN PARAMETERS

In FORECAST/REGRESS, parameters cannot be presented as expressions but only as fields or constants.

(FOC36324) VERB OBJECT OR INTERVAL CANNOT BE A QUOTED CONSTANT

All parameters of FORECAST/REGRESS except 'method' should not contain quotes.

(FOC36325) FORECAST/REGRESS VERB OBJECT IS NOT RECOGNIZED

The verb object is not found in either a Master File or a prior DEFINE section or a COMPUTE statement.

(FOC36330) INTERNAL FORECAST/REGRESS ERROR

Unforeseen error has occurred in routines for FORECAST/REGRESS. Please report the problem to an Information Builders representative.

(FOC36333) INVALID FORECAST/REGRESS REQUEST

'MORE', 'OVER', 'GRAPH', 'MODIFY', 'FOR', and 'BY TOTAL' as well as re-formatting into non-numeric type of field are not permitted in FORECAST/REGRESS.

(FOC36341) INCORRECT USE OF PARAMETERS FOR REGRESSION

The presence of keys equal or preceding the 'ON field' in regression parameter list as well as reusing parameters in the list is not allowed.

NF930: Holding Missing Values

HOLD files can now distinguish between missing data and default values of blank (for character data) or zero (for numeric data).

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 values in a HOLD file. OFF is the default.

Reference Usage Notes for Holding Missing Values

- Setting HOLDMISS ON adds the MISSING=ON attribute to every field in the extract file.
- Data will not be found if:
  - ALL is set to ON.
  - The request is multi-path.
  - The request contains an ACROSS phrase.

Example Holding Missing Values Using HOLDMISS

```
SET HOLDMISS=ON
TABLE FILE MOVIES
 SUM WHOLESALEPR
BY CATEGORY ACROSS RATING
 ON TABLE HOLD AS HLDM
END
TABLE FILE HLDM
 PRINT *
 END
```

The output is:

CATEGORY	WHOLESALEPR	WHOLESALEPR	WHOLESALEPR	WHOLESALEPR	WHOLESALEPR
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	.	.	.

NF931: Multiple FOLD-LINE

You can now specify multiple FOLD-LINE phrases in a TABLE request. Up to sixteen instances of FOLD-LINE may be used in a TABLE request.

Syntax How to Create Multiple FOLD-LINE Clauses in a TABLE Request

`{display|ON|BY} fieldname1... FOLD-LINE fieldname2...`

where:

display

Is any display command.

fieldname2...

Is a field that is moved to the next line because it is referenced after the FOLD-LINE option.

The field name that precedes the FOLD-LINE phrase may be a sort field or display field. If it is a display field, the next field is placed at the start of the next line. If it is a sort field, the next field is on a new line, offset by two spaces from the beginning of the current line.

Example Specifying Multiple FOLD-LINE Phrases in a TABLE Request

```
TABLE FILE MOVIES
SUM WHOLESALEPR LISTPR
BY CATEGORY FOLD-LINE
BY TITLE
BY DIRECTOR FOLD-LINE
IF CATEGORY EQ 'FOREIGN' OR 'DRAMA'
END
```

The output is:

```
CATEGORY
  TITLE                                       DIRECTOR
    WHOLESALEPR  LISTPR
-------------------------------------------------------
DRAMA
  DOG DAY AFTERNOON                           LUMET S.
          10.00   19.98
FOREIGN
  BABETTE'S FEAST                             AXEL G.
          13.25   19.98
  DAS BOOT                                    PETERSON W.
          15.99   29.95
  DEATH IN VENICE                             VISCONTI L.
          30.00   59.95
  FAMILY, THE                                 SCOLA E.
          32.00   59.95
  FANNY AND ALEXANDER                         BERGMAN I.
          25.00   39.98
  TIN DRUM, THE                               SCHLONDORFF V.
          30.00   59.95
```

NF932: Including Data in Multiple FOR Rows in FML

Using the SET FORMULTIPLE command, you can include one input record in multiple tag rows in FML.

Syntax How to Display the Same Record in Multiple FOR Rows

```
SET FORMULTIPLE={ON|OFF}
ON TABLE SET FORMULTIPLE={ON|OFF}
```

where:

`ON`

Allows you to include the same record in multiple FOR rows.

`OFF`

Does not allow you to include the same record in multiple FOR rows. If there are multiple FOR rows present in an FML request for the same record, and FORMULTIPLE is off, the record will only be included in one of the FOR rows. OFF is the default.

Example Displaying the Same Record in Multiple FOR Rows

In the following example, accounts are referenced in more than one row of the request.

```
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
```

NF935: Long Index Names

Names for indexed fields can be up to 66 characters long in an XFOCUS Master File (see *NF916: XFOCUS Data Source*).

Procedure How to Create a Data Source With a Long Index Name

1. Create a Master File with the following attributes:
   - SUFFIX=XFOCUS.
   - An indexed field name longer than 12 characters. Specify FIELDTYPE=I for this field.
2. Issue the CREATE FILE command

   ```
   CREATE FILE name
   ```

   where:

   name

   Is the name of the Master File that specifies the long index name.

Reference Usage Notes for Long Index Names

- If you have an existing XFOCUS data source in which all indexed field names are 12 or fewer characters and you want to change an indexed field name to more than 12 characters, you must:
  1. Issue the REBUILD DUMP command to dump the data source.
  2. Issue the CREATE FILE command, specifying the new Master File.

3. Issue the REBUILD LOAD command to load the data source.

Note: Once at least one indexed field has a name longer than 12 characters, the REBUILD INDEX command can be used to rebuild that index and to index additional fields with names longer than twelve characters.

- If you issue the ? FDT query command, up to twelve characters display for the indexed field name.
- External indexes are not supported.
- Each XFOCUS data source has a File Definition Table (FDT) that contains an entry for each segment and indexed field. The maximum number of entries allowed for segments and indexed fields is 189.

  When this limit is exceeded, the following message is generated:

  ```
  (FOC36347)  UNABLE TO CREATE FOCUS FILE %1: FDT OVERFLOW %2
  ```

  There is a limit to the total number of segments and indices that can exist in a Master. Please check the IBI documentation for exact limits.

  In the message, %1is the name of the Master File and %2 is the number of characters that need to be removed from the total length of the segment and index names.

Example Specifying a Long Index Name in an XFOCUS Master File

```
FILENAME=TEST1, SUFFIX=XFOCUS,
SEGNAME=SEG1, SEGTYPE=S1,$
FIELDNAME=FIELD1             ,ALIAS=FIELD1A ,FORMAT=A20             ,$
FIELDNAME=LONGINDEXFIELDNAME ,ALIAS=LINDFLD ,FORMAT=A10 ,FIELDTYPE=I,$
```

NF936: 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 will become the detail lines in the report. They are not performed on each incoming record as field-based prefix operations are.

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. To display an alphanumeric field on a summary line, you must either explicitly list its 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 on Summary Values

At a sort break, use the following syntax

```
{BY|ON} breakfield [AS 'text1'] summaryoption [MULTILINES]
        operator_and_field_list [AS 'text2'] [WHEN expression;]
```

To replace the default grand total, use the following syntax

```
ON TABLE summaryoption operator_and_field_list [AS 'text2']
```

where:

breakfield

Is the sort field whose change in value triggers the summary operation.

summaryoption

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.

operator_and_field_list

Can have one of the following forms:

[*prefixop.*] *

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 summaryoption for the listed fields. If no field names are listed, the summary is produced for every numeric column in the report output.

prefixop.

The prefix operator is applied to every numeric column in the report output. Every numeric column is populated with values on the summary row.

prefixop1. field1 [field2 ... fieldn] [prefixop2. 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.

where:

prefixop1 ... prefixop2

Are prefix operators for fields used in summary operations. Note: Each prefix operator must be separated from the field name by a space.

field1 ... fieldm

Are the field names to be displayed for the summary line for the specified sort break. Each field name must be preceded by a space.

'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.

- A prefix operator on a summary line must always be followed by a blank space. For example:

  ```
  ON RATING SUBTOTAL AVE. LISTPR
  ```

- Prefix operators PCT., RPCT., AND TOT. are not supported.
- Double prefix operators (such as PCT.CNT.) are not supported.
- The sort field that triggers a summary row cannot be an ACROSS field. An ACROSS field can be used in the request, in which case the same field name will display 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.
- As in the case without prefix operators, different types of summary phrases cannot be combined in one report request.
- 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 (left-most sort field in the output).

Example Using Prefix Operators With SUBTOTAL at the Sort Break Level

The following example uses prefix operators to calculate the:

- Average list price by rating within category.
- Sum of copies by category.

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 default 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 WHOLESALEPR 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 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 WHOLESALEPR 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  WHOLESALEPR  TITLE
------  --------  ------  ------  -----------  -----
G       CHILDREN       2   44.95        29.99  SHAGGY DOG, THE
  .
  .
  .

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 WHOLESALEPR 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  WHOLESALEPR  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 WHOLESALEPR 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  WHOLESALEPR  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

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 will then be applied to any field that does not have a prefix operator.

The new processing is required to display alphanumeric fields on summary lines.

Syntax How to Control Summary Line Processing

```
SET SUMMARYLINES = {OLD|NEW}
```

where:

OLD

Does not allow mixing of summary fields with and without prefix operators when the first field name used in a summary command does not have an associated prefix operator. OLD is the default value. Mixing summary fields with and without prefix operators under these conditions generates the following message

```
(FOC36376) CANNOT COMBINE SUBTOTAL/RECOMPUTE STYLES WHEN
SUMMARYLINES=OLD
```

SUBTOTALs (SUB-TOTAL/RECOMPUTE/SUMMARIZE) with and without prefix operators cannot be combined. To allow the mixture - use SET SUMMARYLINES = NEW.

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

Allows mixing of summary fields with and without prefix operators even when the first field name specified in a summary command does not have a prefix operator. All summary fields without prefix operators are processed as though they were specified with the SUM. operator. Supports display of alphanumeric fields on summary lines.

Reference Usage Notes for SET SUMMARYLINES

- SET SUMMARYLINES is not supported in a TABLE request (ON TABLE).
- 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 will populate only those columns on the associated summary line, while SUMMARYLINES OLD will populate every column specified in any summary command.

  For example:

```
TABLE FILE MOVIES
PRINT COPIES LISTPR WHOLESALEPR
 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. WHOLESALEPR is not referenced in either SUBTOTAL command and, therefore, is not on any summary line:

```
RATING  CATEGORY  COPIES  LISTPR  WHOLESALEPR
------  --------  ------  ------  -----------
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. WHOLESALEPR is not referenced in either SUBTOTAL command and, therefore, is not on any summary line:

```
RATING  CATEGORY  COPIES  LISTPR  WHOLESALEPR
------  --------  ------  ------  -----------
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
```

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 result.

With prefix operator processing, all fields used in the COMPUTE command must be calculated 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 DIFF, 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      4473527        4523963      -50436
             Food            340234      4202338        4183244       19094

*TOTAL West                  696997      8675865        8707207      -31342


TOTAL                       1371188     17192649       17013960      178689
```

The following request uses prefix operators in the RECOMPUTE command to calculate 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      4473527        4523963     -50436
            Food            340234      4202338        4183244      19094

*TOTAL West                             4473527        4183244     290283

TOTAL                                  4473527        4086032     387495
```

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* 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

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 a different prefix operator 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 on the grand total line, unless a different operator is specified for the grand total line.

Example Using Multiple SUB-TOTAL Commands

```
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
```

On the following report output, some of the values have been manually italicized or for clarity:

- The lines preceded by asterisks and the very last (grand total) line are the lines 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 command 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 command is the second summary command and therefore propagates to the DOLLARS column of summary lines for the YEAR sort field.
- Subtotal values in are minimums within their sort groups generated by the command ON REGION SUB-TOTAL MIN. This command 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.

YEAR	State	Region	Category	Unit Sales	Dollar Sales	Budget Dollars
1996	CA	West	Coffee	117539	1,484,873.00	1453548
			Food	116389	1,443,083.00	1414902
			Gifts	74948	947,783.00	946382
*MIN.: West				**74948**	**947,783.00**	**946382**
*AVE. $: CA				**74948**	*1,291,913.00*	**946382**
	IL	Midwest	Coffee	52348	683,559.00	628333
			Food	55867	729,658.00	705915
			Gifts	38405	481,515.00	487090
*MIN.: Midwest				**38405**	**481,515.00**	**487090**
*AVE. $: IL				**38405**	*631,577.33*	**487090**
*CNT. UNITS: 1996				6	*961,745.17*	**487090**
1997	CA	West	Coffee	118044	1,453,023.00	1507092
			Food	106322	1,325,430.00	1302582
			Gifts	77328	988,080.00	961841
*MIN.: West				**77328**	**988,080.00**	**961841**
*AVE. $: CA				**77328**	*1,255,511.00*	**961841**
	IL	Midwest	Coffee	57233	715,220.00	737931
			Food	59293	754,132.00	737912
			Gifts	37918	473,792.00	484611
*MIN.: Midwest				**37918**	**473,792.00**	**484611**
*AVE. $: IL				**37918**	*647,714.67*	**484611**
*CNT. UNITS: 1997				6	*951,612.83*	**484611**
TOTAL				12	*956,679.00*	**484611**

NF937: Optimizing Selection of Relational Variable Length Character Data Types

TABLE IF criteria can now reference RDBMS variable character data types such as VARCHAR, LONG VARCHAR, and CLOB (described in the Master File with USAGE=TX and ACTUAL=TX).

In addition, certain types of IF criteria that reference variable length character data types are included in the generated SQL, causing the selection operations to be performed by the RDBMS and improving performance.

The IF test to be optimized must be a CONTAINS or OMITS test against a field described with USAGE=TX and ACTUAL=TX in the Master File. The RDBMS column must be a character variable length data type.

CONTAINS translates to LIKE in the generated SQL, and OMITS translates to NOT LIKE. The generated SQL places wildcard characters around the literal string specified in the CONTAINS or OMITS test.

Example Optimizing a Selection Test Against a Variable Length Character Column

Consider the following variation of the DB2 Master File named EMPINFO. A CLOB column has been added that contains a job description:

```
FILENAME=EMPINFO ,SUFFIX=SQLDS,$

SEGNAME=EMPINFO  ,SEGTYPE=S0,$
 FIELD=EMP_ID        ,ALIAS=EID         ,USAGE=A9     ,ACTUAL=A9     ,$
 FIELD=LAST_NAME     ,ALIAS=LN          ,USAGE=A15    ,ACTUAL=A15    ,$
 FIELD=FIRST_NAME    ,ALIAS=FN          ,USAGE=A10    ,ACTUAL=A10    ,$
 FIELD=HIRE_DATE     ,ALIAS=HDT         ,USAGE=YMD    ,ACTUAL=DATE   ,$
 FIELD=DEPARTMENT    ,ALIAS=DPT         ,USAGE=A10    ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL        ,USAGE=P9.2   ,ACTUAL=P4     ,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC         ,USAGE=A3     ,ACTUAL=A3     ,$
 FIELD=JOBDESC       ,ALIAS=JDSC        ,USAGE=TX50   ,ACTUAL=TX     ,$
 FIELD=ED_HRS        ,ALIAS=OJT         ,USAGE=F6.2   ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN  ,USAGE=I4     ,ACTUAL=I4     ,$
```

The following request specifies a CONTAINS test against the JOBDESC field:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT

TABLE FILE EMPINFO
PRINT EMP_ID LAST_NAME FIRST_NAME DEPARTMENT CURR_JOBCODE JOBDESC
IF JOBDESC CONTAINS 'PR'
END
```

The CONTAINS operator is translated to a LIKE operator in the generated SQL:

```
    SELECT T1."EID",T1."LN",T1."FN",T1."DPT",T1."CJC",T1."JDSC"
FROM USER1."EMPINFO" T1 WHERE (T1."JDSC" LIKE '%PR%') FOR
FETCH ONLY;
```

Reference Usage Notes for Optimization of Selection of Variable Length Data Types

The following options are not yet supported with text fields:

- CRTFORM
- TYPE
- FSCAN
- MODIFY
- HOLD
- DEFINE
- COMPUTE

NF938: Decimal Alignment of Headings

Numbers on separate lines of a heading, footing, subheading, or subfooting can be displayed so that their decimal points align, even if they have different numbers of decimal places. The decimal alignment works regardless of the type of spacing used by the font, proportional or fixed.

Decimal alignment is supported for styled reports in the following formats:

- HTML.
- PDF.
- PostScript.

Syntax How to Specify Decimal Alignment of a Heading or Footing

To align the decimal points, you define the width of the decimal item, then you 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.

```
TYPE=hcomponent, ITEM=n, WIDTH=w, JUSTIFY=DECIMAL(m), $
```

where:

hcomponent

Specifies the type of heading component that should display with decimal alignment:

`TABHEADING` specifies the report heading.

`TABFOOTING` specifies the report footing.

`HEADING` specifies the page heading.

`FOOTING` specifies the page footing.

`SUBHEAD` specifies a sort heading.

`SUBFOOT` specifies a sort footing.

n

Specifies the part of the heading or footing to align by its position within the heading or footing. Note that several other heading and footing subcomponents can also be used to select a specific item.

w

Specifies the width of the item to be aligned in the units specified by the UNITS parameter (inches by default).

m

Specifies how far in from the right side of a column you want to place your decimal point expressed in the units specified by the UNITS parameter (inches by default). This specification places the decimal point in the same position within a column regardless of the number of decimal places displayed to its right. Text can also be aligned using this technique. The rightmost character aligns in the position immediately to the left of the decimal point.

The measurement must be a portion of the width you have specified for this item.

Example Aligning Data and Text in a Multi-Line Heading or Footing

In the following freeform report, content is defined entirely in the sort heading, where text and data are stacked to support comparison among regions.

The first line of each subheading displays the name of the region. The next three lines display the following numbers decimally aligned. The decimal point displays one inch from the right of a two-inch column:

- Unit sales with no decimal places.
- Dollar sales with two decimal places.
- Budgeted sales with five decimal places.

```
DEFINE FILE GGSALES
JUNITS/D15.0 = UNITS;
JDOLLARS/D15.2 = DOLLARS;
JBUDDOLLARS/D15.5 = BUDDOLLARS;
END
SET SQUEEZE = ON
TABLE FILE GGSALES
BY REGION  NOPRINT SUBHEAD
"SALES FOR: <REGION "
" "
"Units: <JUNITS"
"Dollar Sales: <JDOLLARS"
"Budgeted Sales: <JBUDDOLLARS"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET HTMLCSS ON
ON TABLE SET STYLE *
TYPE = REPORT, GRID=OFF, $
TYPE=SUBHEAD, ITEM=1, WIDTH=1.5                   , $
TYPE=SUBHEAD, ITEM=2, WIDTH=2, JUSTIFY=DECIMAL(1), $
END
```

In order for the second item to align, the first item must have a WIDTH specification so that it will take up a set amount of space. The rightmost character of the region name aligns over the digit to the left of the decimal point in the numeric values. The output is:

```
SALES FOR:                   Midwest

Units:                       905,045.
Dollar Sales:            11,400,665.00
Budgeted Sales:          11,194,373.00000

SALES FOR:                 Northeast

Units:                       916,675.
Dollar Sales:            11,392,310.00
Budgeted Sales:          11,576,932.00000

SALES FOR:                 Southeast

Units:                       935,232.
Dollar Sales:            11,710,379.00
Budgeted Sales:          11,807,981.00000

SALES FOR:                       West

Units:                       932,039.
Dollar Sales:            11,652,957.00
Budgeted Sales:          11,641,513.00000
```

Reference Usage Notes for Decimal Alignment

- To use decimal alignment in an HTML report, you must turn on internal Cascading Style Sheets (SET HTMLCSS=ON).

- Decimal alignment is not supported in requests using ACROSS or OVER.

Aligning Decimals in a Multi-Line Heading or Footing

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 doesn't 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 table:

Floating decimal points	**Aligned decimal points**
Bond Face Value ---------- ---------- Galosh Ltd. 22375.5784596 Mukluk Inc. 1212345.457 Overshoe Inc. 232.45484	Bond Face Value ---------- ---------- Galosh Ltd. 22375.5784596 Mukluk Inc. 1212345.457 Overshoe Inc. 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.

Procedure How to Measure for Column Width and Decimal Alignment

Measuring Width. Determining the width of a heading or footing item is a two-step process:

1. Identify the maximum number of characters in a text string or field.
2. For text strings, simply count the characters. For a field, refer to the format specification in the Master File or in a command such as 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 column, 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.

NF939: SET SAVEDMASTERS

Up to 99 Master Files can be saved in memory after they have been used in a request. Saved Master Files are not re-parsed when referenced in subsequent requests, producing significant performance improvement. The most recently used Master File is stored in memory regardless of this setting. Therefore, this technique is helpful when you run multiple requests against several Master Files.

With each request that specifies a new Master File, the prior Master File is moved down on the saved list and the new Master File is placed at the top of the list. Once all of the slots on the list are full, parsing a new Master File causes the one at the bottom to drop off the list.

If an already saved Master File is used in a request, it moves to the top of the list.

Only one occurrence of a Master File name is maintained on the list. Therefore, if you use an already saved Master File as the host file in a JOIN, in a HOLD command (with the same AS name), in a USE...AS command, or in a COMBINE command without specifying a unique name, the new version of the Master File replaces the previous version on the list. A JOIN CLEAR or USE CLEAR command purges the parsed Master File from memory.

SAVEDMASTERS is not an effective technique to use with massive amounts of data because the amount of time saved by not re-parsing is small in comparison to the time for processing the data.

Syntax How to Save Parsed Master Files in Memory

```
SET SAVEDMASTERS = n
```

where:

n

Is an integer between 0 and 99 that specifies the maximum number of Master Files on the SAVEDMASTERS list. The default value is 0. Note that the most recently used Master File is always stored in memory, even with SAVEDMASTERS set to zero. However, the zero setting does not generate the list of saved Master Files.

Reference Usage Notes for SET SAVEDMASTERS

- Memory resources are used to store the parsed Master Files, reducing the amount of memory available for other processes.
- You cannot selectively purge Master Files from the list.
- The SAVEDMASTERS parameter is not supported in a request (ON TABLE SET) or in FOCPARM.
- The SAVEDMASTERS setting is not supported on a FOCUS Database Server or with a Maintain procedure.
- The SAVEDMASTERS setting is not supported with SCAN or FSCAN.
- Issuing the CHECK FILE or REBUILD command causes the specified Master File to be re-parsed.
- The ?F and ?FF commands only re-parse the Master File when issued outside of a request for a Master File other than the most recently used Master File.

- Using an alternate file view (TABLE FILE *filename.fieldname*) or the AUTOPATH=ON setting re-parses the Master File.
- If the SAVEDMASTERS value is changed between requests:
  - Raising the number allows more Master Files to be saved as they are parsed.
  - Lowering the number drops the oldest saved Master Files.
- If changes are made to a Master File that is saved, the changes will not be implemented until the Master File is re-parsed.
- When only one Master File has been used, it is not placed on the SAVEDMASTERS list.
- DEFINE expressions are not stored and, therefore, are re-parsed every time they are used.
- Creating a HOLD file erases the Master File name from the list if it is there, and the HOLD command does not place the new Master File on the list.
- SAVEDMASTERS is most effective when a Master File has a lot of fields.
- The FML Hierarchy LOAD CHART command does not add the Master File to the SAVEDMASTERS list.

Syntax How to Query the SAVEDMASTERS Setting

The following query command indicates the number of Master Files allowed on the list of saved Master Files and lists the names of the Master Files on the list.

```
? SET SAVEDMASTERS
```

Example Saving and Querying Parsed Master Files

The following command specifies that up to three parsed Master Files can be saved:

```
SET SAVEDMASTERS = 3
? SET SAVEDMASTERS
```

The output of the query command indicates that the list can contain up to three Master Files, but none are currently saved:

```
SAVEDMASTERS                3
```

The following procedure parses two Master Files, EMPLOYEE and MOVIES:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME BY EMP_ID
END
-RUN
TABLE FILE MOVIES
PRINT TITLE BY DIRECTOR
END
-RUN
? SET SAVEDMASTERS
```

The output of the query command indicates that the list can contain up to three Master Files and that the list currently consists of MOVIES and EMPLOYEE:

```
SAVEDMASTERS                3

  MOVIES
  EMPLOYEE
```

NF940: SET HNODATA

The SET HNODATA command controls the missing data characters that are propagated to fields with the MISSING=ON attribute in HOLD FORMAT ALPHA files. Missing values in fields that do not have the MISSING=ON attribute are propagated to a HOLD file as blank (for alphanumeric fields) or zero (for numeric fields).

Syntax How to Control Missing Values Propagated to a HOLD File

```
SET HNODATA = {charstring|,$}
```

In a TABLE request:

```
ON TABLE SET HNODATA {charstring|,$}
```

where:

charstring

Is a string of up to 12 characters that will be propagated to a HOLD FORMAT ALPHA file for missing values in a field with the MISSING=ON attribute. The default value is a period (.).

If the string is longer than the length of the field, the value stored in:

- An alphanumeric field will be the leftmost characters of the string.
- A numeric field will be a blank string.

When an alphanumeric string other than the default value (the period) is used to populate a missing numeric field, a blank will be inserted in the held field to prevent a format error when displaying the data. If you use the default HNODATA value, it is inserted in numeric fields. In this way, a request against the HOLD file can recognize missing data that was propagated to the HOLD file.

If a number with decimal places is specified for HNODATA and the field with missing data is integer, the value is rounded to a whole number and then inserted. In a numeric field that supports decimal places, it will be rounded and inserted with the correct number of decimal digits.

`,$`

Indicates that nothing should be placed in the field when there is missing data. This setting can be used to support null values in non-FOCUS data sources.

Example Controlling Missing Data Values in an Alphanumeric HOLD File

First edit the BANK_NAME and BANK_CODE fields in the EMPLOYEE Master File to support missing data:

```
FIELDNAME=BANK_NAME,    ALIAS=BN,      FORMAT=A20, MISSING=ON, $
FIELDNAME=BANK_CODE,    ALIAS=BC,      FORMAT=I6S, MISSING=ON, $
```

The following request sets the HNODATA string to the string 'N/A' and creates an alphanumeric HOLD file named HMISS:

```
SET HNODATA = 'N/A'

TABLE FILE EMPLOYEE
PRINT BANK_NAME BANK_CODE
BY EMP_ID
ON TABLE HOLD FORMAT ALPHA AS HMISS
END
-RUN

TABLE FILE HMISS
PRINT *
END
```

Note that a blank is inserted in the integer field BANK_CODE:

```
EMP_ID      BANK_NAME             BANK_CODE
------      ---------             ---------
071382660   N/A
112847612   N/A
117593129   STATE                    510271
119265415   N/A
119329144   BEST BANK                302161
123764317   ASSOCIATED               112600
126724188   N/A
219984371   N/A
326179357   ASSOCIATED               112600
451123478   ASSOCIATED               112600
543729165   N/A
818692173   BANK ASSOCIATION         175963
```

Example Propagating Numeric Values for Missing Data to a HOLD File

First edit the BANK_NAME and BANK_CODE fields in the EMPLOYEE Master File to support missing data:

```
FIELDNAME=BANK_NAME,    ALIAS=BN,      FORMAT=A20, MISSING=ON, $
FIELDNAME=BANK_CODE,    ALIAS=BC,      FORMAT=I6S, MISSING=ON, $
SET HNODATA = 123.65
TABLE FILE EMPLOYEE
PRINT BANK_NAME BANK_CODE
BY EMP_ID
ON TABLE HOLD FORMAT ALPHA AS HMISS
END
-RUN

TABLE FILE HMISS
PRINT *
END
```

The number 123.65 is rounded to 124 before being inserted into the integer BANK_CODE field:

```
EMP_ID      BANK_NAME              BANK_CODE
------      ---------              ---------
071382660   123.65                       124
112847612   123.65                       124
117593129   STATE                     510271
119265415   123.65                       124
119329144   BEST BANK                302161
123764317   ASSOCIATED               112600
126724188   123.65                       124
219984371   123.65                       124
326179357   ASSOCIATED               112600
451123478   ASSOCIATED               112600
543729165   123.65                       124
818692173   BANK ASSOCIATION         175963
```

Example Creating a COMT Data Source With Null Data

First edit the BANK_NAME and BANK_CODE fields in the EMPLOYEE Master File to support missing data:

```
FIELDNAME=BANK_NAME,    ALIAS=BN,      FORMAT=A20, MISSING=ON, $
FIELDNAME=BANK_CODE,    ALIAS=BC,      FORMAT=I6S, MISSING=ON, $
```

```
SET HNODATA = ,$

TABLE FILE EMPLOYEE
PRINT BANK_NAME BANK_CODE  LAST_NAME
BY EMP_ID
ON TABLE HOLD FORMAT COMT  AS HMISS
END
-RUN

TABLE FILE HMISS
PRINT *
END
```

The output is:

```
EMP_ID     BANK_NAME             BANK_CODE  LAST_NAME
------     ---------             ---------  ---------
071382660                                    STEVENS
112847612                                    SMITH
117593129  STATE                    510271  JONES
119265415                                    SMITH
119329144  BEST BANK                302161  BANNING
123764317  ASSOCIATED               112600  IRVING
126724188                                    ROMANS
219984371                                    MCCOY
326179357  ASSOCIATED               112600  BLACKWOOD
451123478  ASSOCIATED               112600  MCKNIGHT
543729165                                    GREENSPAN
818692173  BANK ASSOCIATION         175963  CROSS
```

The COMT file created contains null values for the missing data. In the alphanumeric BANK_NAME field, two consecutive double quotation marks are placed in the field. In the integer BANK_CODE field, nothing is placed in the field. This is indicated by two consecutive commas:

```
"EMP_ID","BANK_NAME","BANK_CODE","LAST_NAME"
"071382660","",,"STEVENS"
"112847612","",,"SMITH"
"117593129","STATE",510271,"JONES"
"119265415","",,"SMITH"
"119329144","BEST BANK",302161,"BANNING"
"123764317","ASSOCIATED",112600,"IRVING"
"126724188","",,"ROMANS"
"219984371","",,"MCCOY"
"326179357","ASSOCIATED",112600,"BLACKWOOD"
"451123478","ASSOCIATED",112600,"MCKNIGHT"
"543729165","",,"GREENSPAN"
"818692173","BANK ASSOCIATION",175963,"CROSS"
```

NF943: Increased Number of ACROSS Columns

The maximum number of report columns created using the ACROSS phrase in a report is 1056.

The actual number of columns created by ACROSS depends on the number of fields being displayed, the number of values for each of these fields, and the number of ACROSS and ACROSS-TOTAL phrases in the request.

If you exceed the maximum number of ACROSS columns, the following message displays:

```
(FOC153)     VERB OBJECTS TIMES RETRIEVED COLUMNS EXCEEDS MAXIMUM
```

The combination of ACROSS and the number of verb objects will produce more than the maximum number of output columns allowed
in the report. Reduce the number of verb objects. See the documentation for further details.

Reference Usage Notes for Number of ACROSS Columns

- The sizes of the fields being displayed in the report also affect the number of columns allowed because 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.
- The sizes of column titles created using the AS phrase also affect the number of columns allowed in the report.

NF944: Increased Space for Column Titles

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.

NF945: Creating Compound PDF or PostScript 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} [NOBREAK]
```

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 [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 [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 species 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 displays 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 display on a separate page.)

Sales Report

Product Cateogory:	Line Total
Camcorders	$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

Product Cateogory:	Quantity:
Camcorders	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

Product Cateogory:	Line Cost Of Goods Sold
Camcorders	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

NF946: &FOCUSER

The variable &FOCUSER retrieves the user ID of the connected user. It retrieves the same value as the GETUSER subroutine without the overhead of a subroutine.

Example Displaying the Connected User

```
-TYPE &FOCUSER
```

The output is:

```
USERID1
```

(If USERID1 is the connected ID)

Reference Usage Notes for &FOCUSER

- &FOCUSER is a system variable that cannot be set or changed by the user.
- If you have previously defined a variable as &FOCUSER in any of your procedures the previous value will be overwritten by the value of the system variable &FOCUSER.

NF951: Defining and Displaying FML Hierarchies

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 numeric or both alphanumeric.

For example, suppose that:

- An employee data source contains both the employee's ID and the ID of the employee's manager.

  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. However, to print the chart in a traditional FML report, you must 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
 .
 .
 .
```

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.
2. The hierarchy must be loaded into memory. The loaded hierarchy is called a *chart*. If the hierarchy is defined in the Master File and referenced by the FML request, the hierarchy is loaded automatically. If you want to use a hierarchy defined in a Master File that will not be either referenced in the FML request or joined to the Master File referenced in the FML request, you must 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.
3. 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* it must be linked to a parent-child hierarchy. 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.

Syntax How to Specify a Hierarchy Between Fields in a Master File

```
FIELD=parentfield,..., PROPERTY=PARENT_OF,
REFERENCE=[seg.]hierarchyfld,$
```

where:

parentfield

Is the parent field in the hierarchy.

`PROPERTY=PARENT_OF`

Identifies this field as the parent of the referenced field in a hierarchy.

These attributes can be specified on every field. Therefore, multiple hierarchies can be defined in one Master File. However, an individual field can have only one parent. If multiple fields have PARENT_OF attributes for the same hierarchy field, the first parent found by traversing the structure in top-down, left-to-right order is used as the parent.

seg

Is the segment location of the hierarchy field. Required if more than one segment has a field named *hierarchyfld*.

hierarchyfld

Is the child field in the hierarchy.

PARENT_OF is also allowed on a virtual field in the Master File:

`DEFINE name/fmt=expression;,PROPERTY=PARENT_OF,REFERENCE=hierarchyfld ,$`

Syntax How to Assign Descriptive Captions for Hierarchy Field Values

The following attributes specify a caption for a hierarchy field in a Master File

`FIELD=captionield, ..., PROPERTY=CAPTION, REFERENCE=[seg.]hierarchyfld, $`

where:

captionfield

Is the name of the field that contains the descriptive text for the hierarchy field. For example, if the employee ID is the hierarchy field, the last name may be the descriptive text to be displayed on the report in place of the ID.

`PROPERTY=CAPTION`

Signifies that this field contains a descriptive caption to be displayed in place of the hierarchy field values.

A caption can be specified for every field, but an individual field can have only one caption. If multiple fields have CAPTION attributes for the same hierarchy field, the first parent found by traversing the structure in top-down, left-to-right order will be used as the caption.

seg

Is the segment location of the hierarchy field. Required if more than one segment has a field named *hierarchyfld*.

hierarchyfld

Is the hierarchy field.

CAPTION is also allowed on a virtual field in the Master File:

```
DEFINE name/fmt=expression;,PROPERTY=CAPTION,REFERENCE=hierarchyfld ,$
```

Example Defining a Hierarchy in a Master File

The 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:

```
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 century corp financial data. 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', $
FIELDNAME = NAT_YTDBUD    ,  ,D12.0    , TITLE='YTD,Budget', $
```

Displaying 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.

When a request references a hierarchy, an internal TABLE request is automatically run to produce the 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 Display an FML Hierarchy

```
TABLE FILE filename
{PRINT|SUM} ....
FOR hierarchyfld
parentvalue {GET|WITH} CHILD[REN] [n|ALL] [AS CAPTION|'text'] [LABEL lab]
.
.
.
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, you must have previously issued the LOAD CHART command to load the hierarchy.)

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*. 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*. 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, 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 next child instance. 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: 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 will not be used as the caption on the report output.

lab

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.

Example Displaying an FML Hierarchy

The following request displays two levels of account numbers starting from account 3000:

```
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 had specified GET CHILDREN instead of WITH CHILDREN, the line for the parent value (3000) would not have displayed on the report output.

Example Displaying an FML Hierarchy With Captions

The following request displays two levels of 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 had specified GET CHILDREN instead of WITH CHILDREN, the line for the parent value (1000, Profit Before Tax) would not have displayed on the report output.

Consolidating an FML Hierarchy

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 will have a line in the report output and the summary for each child will include 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 print 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, you must have previously issued the LOAD CHART command to load the hierarchy.)

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, direct children are included. If *n* is 2, direct children and grandchildren are included. ADD 99 included children to 99 levels. 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 will display for the total row.

Note: 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 aggregate row. The CAPTION field defined in the Master File will not be 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: The join is not required to be unique because the hierarchy is defined in the host segment.

First the lines of the hierarchy starting with account 3100 (Selling Expenses) are displayed by the WITH CHILDREN command. 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          YTD
                                        Actual         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/Commisions                     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, you must have previously issued the LOAD CHART command to load the hierarchy.)

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 children to display. The line of output for each child will have 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, direct children each have 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

For each line generated by the GET or WITH CHILDREN command, sums the hierarchy to the depth specified by *m*.

m|ALL

Is a positive integer from 1 to 99 specifying the number of levels 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 will sum all children. If *m* is 2, direct children and grandchildren will be 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 will display for the total row.

Note: 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 aggregate row. The CAPTION field defined in the Master File will not be 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: 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 had had multiple levels of children, they would all have been added into the sum. The other direct children of 3100 did not themselves have children, so the sum on each of those lines consists of only the parent value.

```
                                        Month         YTD
                                        Actual        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/Commisions                    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/Commisions                     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          YTD
                                        Actual         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/Commisions                      100,188.       304,199.

                                    ============   ============

Advertising                           1,303,277.     3,705,368.
Promotional Expenses                     53,719.       151,732.
Joint Marketing                          97,135.       289,799.
Bonuses/Commisions                      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          YTD
                                          Actual         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

In most cases, the 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 16. Charts are automatically unloaded when the session ends.

Syntax How to Load a Hierarchy From One Master File for Use With a Separate Master File

LOAD CHART *chartfile*[.*sega*].*hierarchyfld* [FOR *requestfl*[[.*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. The hierarchy field is required because a Master File can define multiple hierarchies.

FOR

Loads a hierarchy defined in a Master File that will not be used in the FML report request. For example, Master File B contains the hierarchy information but Master File A will be used in the request (without a join between Master Files A and B), you would issue the following LOAD CHART command prior to the FML request:

```
LOAD CHART B.FLDB FOR A.FLDA
TABLE FILE A ...
```

requestfl

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 depending on the value of the HOLDATTR parameter.
- 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 is 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 will be counted twice in an aggregation operation. To avoid this double aggregation, use the FST. operator in the SUM command for the shared fields.

Reference FML Hierarchy Error Messages

LOAD CHART messages:

(FOC36361) **REQUESTED HIERARCHY NOT FOUND FOR MASTER.**

This message occurs at LOAD CHART time when the Master File cannot be found, or the hierarchy within the Master cannot be found.

(FOC36362) **NOT ENOUGH MEMORY TO LOAD REQUESTED HIERARCHY.**

This message occurs when there is not enough available memory to load the requested hierarchy.

(FOC36363) **NO DATA FOUND FOR REQUESTED HIERARCHY.**

The hierarchy does not have any data associated with it.

TABLE request messages:

(FOC36364) **REQUESTED HIERARCHY IS NOT LOADED.**

This message occurs when a hierarchy is needed to satisfy the request, but that hierarchy is not loaded.

(FOC36365) **SYNTAX ERROR IN FML HIERARCHY COMMAND**

Only GET CHILDREN, ADD, and WITH are allowed. All of these must immediately follow a field value; these cannot be used with an OR-phrase, TO-phrase, or masking tags

(FOC36366) **WRONG NUMERIC VALUE IN GET/ADD/WITH CHILD n PHRASE**

Numeric value should be a positive integer less than 99. In case n is greater than 99, it is set to be equal to 99.

NF952: TABLASTPAGE

TABLASTPAGE is a system variable that contains the total number of pages in a report. It can be used in conjunction with the page number system variable, TABPAGENO, in a heading, footing, subhead, or subfoot to print "PAGE X of Y" on a PDF, Postscript, or HTML report.

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:

Salary	Percent Increase	Date of Increase	Employee ID
$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

Salary	Percent Increase	Date of Increase	Employee ID
$10,000.00	.10	82/01/01	071382660
			112847612
	.12	81/01/01	071382660

Page 2 of 5

NF953: File Extensions For Maintain Procedures

Maintain procedures can have file extensions different from other FOCEXECs. For MVS, the ddname is MAINTAIN and for CMS the file type is MAINTAIN.

Syntax How to Work With Maintain File Extensions

In order to work with Maintain procedures that have their own extensions, the MNTCON command must be used as follows:

To execute a Maintain procedure:

```
MNTCON EX maintproc
```

To compile a Maintain procedure:

```
MNTCON COMPILE maintproc
```

This creates a compiled procedure with the filetype FOCCOMP on CMS (ddname FOCCOMP should be allocated for MVS).

To run a compiled Maintain procedure:

```
MNTCON RUN maintproc
```

where:

maintproc
: Is a compiled Maintain procedure.

Reference Usage Notes for Maintain File Extensions

If Maintain does not find the procedure under file type MAINTAIN on VM or ddname MAINTAIN on MVS, it looks for file type FOCEXEC on VM or ddname FOCEXEC on MVS.

Example Defining File Extensions for Maintain

In MVS JCL:

```
//MAINTAIN DD=myqual.MAINTAIN.DATA.DISP=SHR
```

In VM:

```
test1 MAINTAIN A1
```

NF954: Enhanced Screening Conditions for Maintain

You can screen incoming field values by checking them against a list of values or mask in Maintain. You also can select records that contain or omit an alphanumeric string.

Syntax How to Use the WHERE Clause in a Maintain Procedure

The following WHERE operators have been added for use in a Maintain procedure:

`NOT_IN` *`list ('item1', 'item2' [,...])`*

Selects records with values that are not in the list. It is the converse of IN.

Spaces or commas can separate the list of values. List values need not be enclosed in single quotation marks unless their value contains a space.

`EQ_MASK`

Selects records with values that match a mask.

Use the $ sign to replace each letter in the value. Masks can only be used with alphanumeric data. The masked value may be a literal or a variable.

`NE_MASK`

Selects records with values that do not match a mask.

Use the $ sign to replace each letter in the value. Masks can only be used with alphanumeric data. The masked value may be a literal or a variable.

`CONTAINS` *`string`*

Selects records with values that contain the alphanumeric string.

`OMITS string`

Selects records with values that do not contain the alphanumeric string.

Example Using NOT_IN List With Maintain

The following example shows how to load a stack with records whose values are not in a specified list:

```
FOR ALL NEXT EMP_ID BANK_NAME INTO STACK1
WHERE BANK_NAME NOT_IN ('ASSOCIATED BANK', CITIBANK)
```

This returns all data where the bank name is not ASSOCIATED BANK or CITIBANK. NOT_IN can be used with all data types.

Example Using EQ_MASK With Maintain

The following example shows how to load a stack with records whose values match a mask:

```
COMPUTE CODE/A4='AAA$';
FOR ALL NEXT BANK_NAME INTO STACK1
    WHERE BANK_NCODE EQ_MASK CODE;
```

This returns all data where the bank code starts with AAA.

Conversely, the code:

```
COMPUTE CODE/A4='AAA$';
FOR ALL NEXT BANK_NAME INTO STACK1
    WHERE BANK_CODE NE_MASK CODE;
```

Returns all data where the bank code doesn't match the mask.

Example Using CONTAINS and OMITS With Maintain

The following example shows how to load a stack with records that contain an entered string:

```
COMPUTE NAME/A4 = 'BANK';
FOR ALL NEXT BANK_CODE BANK_NAME INTO STACK1
    WHERE BANK_NAME CONTAINS NAME;
```

This returns all data where the word BANK is part of the bank name.

Conversely, the code:

```
COMPUTE NAME/A4 = 'BANK';
FOR ALL NEXT BANK_CODE BANK_NAME INTO STACK1
    WHERE BANK_NAME OMITS NAME;
```

This returns all data where the bank name does not include the word BANK.

NF955: Long Qualified Field Names

A qualified field name consists of a file name, a segment name, and a field name, separated by QUALCHARs (qualification characters).

File names can be up to 64 characters in length. Segment names can be up to 8 characters in length in FOCUS data sources or 64 characters in length in non-FOCUS or XFOCUS data sources. Field names can be up to 66 characters. Therefore, a qualified field can have a total of 138 characters in a FOCUS data source and a total of 194 characters in a non-FOCUS or XFOCUS data source, plus the two qualification characters.

Because of the two qualification characters (usually displayed as two dots) included with every qualified field name, the total length of a long qualified field name, can be up to 140 characters in a FOCUS data source and up to 196 characters in a non-FOCUS or XFOCUS data source.

Long qualified field names are useful for the storage of documents that use large descriptive field names such as those that are used and transmitted with XML protocol.

Reference Usage Notes for Long Qualified Field Names

- Long qualified field names can be used with MVS, UNIX, and Windows® NT.
- FIELDNAME must be set to NEW.
- Long field names can be coded within the Master File or in a DEFINE.

Example How to Use Long Qualified Field Names

Sample Master File:

```
FILENAME=INTSALES,SUFFIX=FOC
SEGNAME=FRANCE,SEGTYPE=S1
FIELDNAME=PROVINCE_NICE_STORE500_TOTALSALES,TSALE500,A10,$
FIELDNAME=PROVINCE_NICE_STORE501_TOTALSALES,TSALE501,A10,$
```

Sample FOCEXEC:

Note: For display purposes, the FOCEXEC contains CREATE FILE and MODIFY commands.

```
DYNAM ALLOC DD INTSALES LONGNAME -
 AVANT_GARDE_CLOTHING_DATASOURCE -
 DS PMSESD.INTSALES.FOCUS SHR REU
 -RUN
 -*
 CREATE FILE
 AVANT_GARDE_CLOTHING_DATASOURCE
 -RUN
 MODIFY
 FILE AVANT_GARDE_CLOTHING_DATASOURCE
 FREEFORM TSALE500 TSALE501
 MATCH TSALE500
 ON MATCH REJECT
 ON NOMATCH INCLUDE
 DATA
 12235,5470,$
 9827,7675,$
 7635,9327,$
10325,7543,$
END
-RUN
TABLE FILE AVANT_GARDE_CLOTHING_DATASOURCE
PRINT
AVANT_GARDE_CLOTHING_DATASOURCE.FRANCE.PROVINCE_NICE_STORE500_TOTALSALES
AVANT_GARDE_CLOTHING_DATASOURCE.FRANCE.PROVINCE_NICE_STORE501_TOTALSALES
END
-RUN
```

The output is:

```
PROVINCE_NICE_STORE500_TOTALSALES  PROVINCE_NICE_STORE501_TOTALSALES
---------------------------------  ---------------------------------
12235                              5470
9827                               7675
7635                               9327
10325                              7543
```

Reference Error Messages for Long Qualified Field Names

The following error message is displayed:

```
(FOC003) THE FIELDNAME IS NOT RECOGNIZED: fieldname
               BYPASSING TO END OF COMMAND
```

when either:

- FIELDNAME is set to OLD while using a long qualified field name.
- A long file name exceeds 64 characters in length.
- A segment name exceeds 8 characters in length in a FOCUS file, or 64 characters n a relational or XFOCUS file.
- A long field name exceeds 66 characters in length.
- Any combination of a long file name, a segment name or long segment name, and a long field name exceeds 140 characters in length in a FOCUS file or 196 characters in length in a relational or XFOCUS file.

NF956: Wide Lines

The FOCEXEC, MASTER, ACCESS, WINFORMS, and MAINTAIN ddnames on MVS or file types on VM can be stored as variable length files with LRECL up to 32K. HOLDMAST can also be stored with wide lines on MVS.

Wide lines can make coding more convenient. For example, you can:

- Describe all of the attributes of a field on one line in a Master File.
- Specify an entire SQL statement on one line.
- Place all TABLE syntax on one line, except for those commands that must be on their own line (for example, END).
- Avoid confusing continuation lines that require commas or dashes at the end or dashes at the beginning.
- Include an entire FIXFORM or CRTFORM statement on one line of a FOCEXEC.
- Issue EX commands with many arguments on one line of a FOCEXEC.

On MVS, a wide file can be a member of a PDS or can be stored in a sequential file. You can concatenate these data sets with data sets that have RECFM=F and LRECL=80, in any order. Any new file you create will be written to the first data set in the concatenation. If you do not have write access to this data set, you will get a security error.

The FOCUS text editor, TED, cannot be used to edit wide files. For information on editing files with wide lines, see *NF984: IEDIT Command*.

Reference Usage Notes for Wide Lines

- The maximum LRECL for a variable-length file is 32756 bytes (32760 minus a four-byte overhead to store the actual length). For fixed length the maximum is 32760.
- You can allocate, create, and use wide files with the same techniques that apply to standard files. Only the DCB parameters are different. To edit wide files you can use IEDIT, a new facility distributed with FOCUS for invoking the system editor (XEDIT on VM or ISPF EDIT on MVS) with some added functionality. You can also use your system editor directly. TED does not support wide files.
- There is no performance advantage to converting existing RECFM F LRECL 80 files to wide format.
- FOCUS creates HOLD Master Files as RECFM F LRECL 80. If you create a HOLD file from a data source with a TITLE attribute longer than 80 characters and if HOLDATTRS=ON, the TITLE attribute will be truncated in the HOLD Master File.
- Because of the nature of Mainframe retrieval of variable-length records, certain applications may be subject to more overhead with wide files than with fixed length files. To minimize this effect, allocate PDS libraries containing wide files with large block sizes in order to improve retrieval speed.
- Blanks at the end of a line may be retained when you use continuation lines in a heading or with certain commands, such as -SET and CRTFORM.
- The USE command, -INCLUDE command, EX command, and the FOCUS Access File support mixing of wide and standard files.

- **Tip:** With variable length records, FOCUS has to search for the end of each line to find the label that indicates the target of a -GOTO or the end of a -REPEAT. Therefore, removing loops from a wide FOCEXEC and replacing them with multiple EX or -INCLUDE commands may speed up execution.
- If you try to access a wide PDS in a release prior to FOCUS 7.3, an error message will result.

Reference Messages for Wide Lines

The following error message is issued when the LRECL of a library exceeds 32760:

(FOC900) **%1 LIBRARY *lib* EXCEEDS LRECL LIMIT OF 32760 (DDNAME=*ddn*)**

Maximum record length of a library cannot exceed the value shown in the message. All libraries within the ddname cannot be accessed.

The following message is issued when a line is too long to be written to a library:

(FOC901) **A *nn* BYTE LINE IS TOO LONG TO BE WRITTEN TO *lib* LIBRARY**

Length error: the line cannot be written to the library because its length exceeds the library LRECL. The library must be allocated with LRECL not less than the length of the line being written.

The following message is issued when the first data set name in a concatenation is RECFM=F and a subsequent data set in the concatenation is a PDSE with RECFM=V:

(FOC903) **INVALID %1 LIBRARY CONCATENATION (DDNAME=%2 DSN=%3)**

The dataset mentioned in the message is a PDSE with recfm=V, but it belongs to the concatenated DD where the first dataset has recfm=F. This combination is not allowed. All libraries within the ddname cannot be accessed

NF957: FIELDTYPE=R

The field-level Master File attribute FIELDTYPE=R (Read-only) identifies relational auto-increment columns such as timestamp columns in DB2. These columns are automatically incremented by the RDBMS. Any explicit command to update this type of column is ignored.

Note: This attribute is only supported for DB2 on MVS with TIMESTAMP columns.

Syntax How to Specify a Field as Read-Only in a Master File

```
FIELD= ..., FIELDTYPE = R,$
```

where:

`R`

Indicates that the field is read-only. Any number of fields can have this attribute. The field with this attribute must represent a DB2 TIMESTAMP column.

Reference Usage Notes for FIELDTYPE=R

AUTODB2 does not add the FIELDTYPE=R attribute to the FIELD declaration in the generated Master File. You must edit the Master File to add this attribute.

Example Using FIELDTYPE=R

SALINFO is a DB2 table with a TIMESTAMP column (field ENTRY_DATE in the Master File):

```
FILENAME=SALINFO,  SUFFIX=SQLDS,$

SEGNAME=SALINFO  ,SEGTYPE=S0,$
 FIELDNAME=SALEID    ,ALIAS=EID     ,USAGE=A9      ,ACTUAL=A9,$
 FIELDNAME=PAY_DATE  ,ALIAS=PD      ,USAGE=YYMD    ,ACTUAL=DATE,$
 FIELDNAME=GROSS     ,ALIAS=MO_PAY  ,USAGE=D12.2M  ,ACTUAL=D8,$
 FIELDNAME=ENTRY_DATE,ALIAS=EDATE   ,USAGE=HYYMDm  ,ACTUAL=HYYMDm,
 FIELDTYPE=R,$
```

The following MODIFY procedure asks for an employee ID and pay date. If the employee ID exists in the table, but the pay date does not, the procedure asks for a monthly salary and adds a new row to the table:

```
MODIFY FILE SALINFO
CRTFORM LINE 1
" PLEASE ENTER A VALID EMPLOYEE ID AND PAY_DATE </1"
" EMPLOYEE ID: <SALEID   PAY DATE: <PAY_DATE "
MATCH SALEID
ON NOMATCH GOTO EXIT
ON MATCH GOTO ADDCASE
CASE ADDCASE
MATCH PAY_DATE
ON MATCH GOTO EXIT
ON NOMATCH CRTFORM LINE 5
" ENTER NEW MONTHLY SALARY <T.GROSS>  "
ON NOMATCH INCLUDE
ENDCASE
DATA
END
```

The following rows exist in the table for employee ID 071382660:

```
SALEID      PAY_DATE              GROSS  ENTRY_DATE
------      --------              -----  ----------
071382660   2001/11/30        $5,000.33  2001/11/30 12:12:28.625314
071382660   2001/12/31        $5,000.33  2001/12/31 12:12:42.610556
071382660   2002/01/29        $5,083.67  2002/01/29 12:13:26.823951
071382660   2002/02/26        $5,083.67  2002/02/26 12:13:43.376119
071382660   2002/03/29        $5,083.67  2002/03/29 12:14:03.218765
071382660   2002/04/30        $5,083.67  2002/04/30 12:14:16.018353
071382660   2002/05/28        $5,083.67  2002/05/28 12:14:28.918522
071382660   2002/06/28        $5,083.67  2002/06/28 12:14:42.759658
071382660   2002/07/30        $5,083.67  2002/07/30 12:14:55.088765
071382660   2002/08/30        $5,083.67  2002/08/30 12:15:10.717233
```

The following example executes the MODIFY procedure to add a salary of $6,000 for the pay date 03/04/30 for this employee:

```
 PLEASE ENTER A VALID EMPLOYEE ID AND PAY_DATE

 EMPLOYEE ID:  071382660    PAY DATE:  20030430

 ENTER NEW MONTHLY SALARY  6000
```

The RDBMS automatically generates a timestamp value for the ENTRY_DATE field. If the procedure had supplied a value, it would have been ignored:

```
SALEID      PAY_DATE              GROSS  ENTRY_DATE
------      --------              -----  ----------
071382660   2003/04/30        $6,000.00  2003/05/12 14:43:17.852050
```

It is up to the developer to query the data source to find out what value was generated by the RDBMS for the auto-increment field.

NF958: StyleSheet Enhancements

A styled report is a report produced as format HTML, PDF, PostScript, Excel 2000, or Excel 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.

This bulletin contains detailed information on StyleSheet enhancements as well as all current information about styling reports.

Introduction to Styled Reports

A StyleSheet enables you to format and produce attractive reports that highlight key information.You can use a StyleSheet to:

- Format report components individually.
- Incorporate graphical elements.
- Format data that meets specified conditions.
- Create macros that enable you to streamline your formatting specifications.

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 of which identifies the report component, such as a heading, column, or grand total, whose characteristics you wish to define, and describes the desired characteristics of that report component. 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 *NF973: Cascading Style Sheets*.

Unless otherwise noted, all StyleSheet references in this document refer to FOCUS StyleSheets.

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 can 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, it will override the orange and be displayed as green. For more information, see *FOCUS StyleSheet Attribute Inheritance*.

- **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.

  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.

- **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.

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:

Century Corporation

Order Revenue - 2001 Q3

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. "C e n t u r y C o r p o r a t i o n"
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 STYLESHEET 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's margins instead of the page'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'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's column titles.
9. Formats the report'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.

How to Choose 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

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 via 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's HTML formatting for that report. The resulting report displays in a fixed font without colors and other Web capabilities.

- **Print formats—PDF** (Adobe® Acrobat's® 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 via 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.

  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*.

- **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 the following formats: Excel 2000 HTML-based format, with variations for Excel 2000 PivotTable and Excel 2000 FORMULA; 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.

  - **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 *NF888: Excel 2000* and *NF989: Excel 97*.

    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 MVS, 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

FOCUS StyleSheet	File/Member Name
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 fuschia backgrounds for selected components	FUSCHIA
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

FOCUS StyleSheet	File/Member Name
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

The process of using a StyleSheet consists of the following steps:

1. Decide which StyleSheet to use. If one already exists with the formatting you need, go on to the next step. FOCUS comes with default styles that you can use if you want the whole report printed with the same default format. If you want to customize certain formats in your report, create a StyleSheet that describes those formats.

   Note: 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 describing 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.

If you do not create a StyleSheet, FOCUS uses the same default style for all report components.

If you create a StyleSheet, you only have to define the styles of those components that you want displayed differently from the default style. Any component that you do not specifically format in your StyleSheet either uses the default style or inherits a style from a higher-level component. Inheritance is discussed in *FOCUS StyleSheet Attribute Inheritance*.

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?

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 what you can do with and without StyleSheets:

	With StyleSheets	**Without StyleSheets**
Text font	You can use different font sizes and fonts. You can selectively apply text styles such as bold or 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.

	With StyleSheets	Without StyleSheets
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.

	With StyleSheets	Without StyleSheets
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 and PDF reports.	FOCUS uses the default orientation specified in your printer setup.
Page margins	You can specify the top, bottom, left, and right margins, measured in inches, centimeters, or points 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

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*.

- 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*.

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 is highlighted.

```
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

DEPARTMENT	EMP_ID	FIRST_NAME	LAST_NAME
MIS	112847612	MARY	SMITH
	117593129	DIANE	JONES
	219984371	JOHN	MCCOY
	326179357	ROSEMARIE	BLACKWOOD
	543729165	MARY	GREENSPAN
	818693173	BARBARA	CROSS
PRODUCTION	071382660	ALFRED	STEVENS
	119265415	RICHARD	SMITH
	191399144	JOHN	BANNING
	223754317	JOAN	IRVING
	267191168	ANTHONY	ROMANS
	451299478	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*.

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 you want your report to look. Each declaration:

- Identifies a report component or subcomponent.
- Describes the formatting to apply to that component.
- Optionally, if the component is a heading, footing, or column, specifies a position on the page for the component 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 you want to change. Within each declaration, include only those formatting attributes that you want 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* 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 terminating dollar sign. 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

You can check the syntax of a StyleSheet from the FOCUS prompt with the CHECK STYLE command.

Syntax How to Check StyleSheet Syntax

```
CHECK STYLE filename
```

where:

filename

Is the name of the StyleSheet file.

FOCUS reports any syntax errors in the StyleSheet file. It does not check whether the specified fonts are available or whether the font names are spelled correctly.

Creating 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 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

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
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.

Value	Description
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 or PS report.

```
[TYPE=REPORT,] ORIENTATION={PORTRAIT|LANDSCAPE}, $
```

where:

TYPE=REPORT

Applies the page orientation to the entire report. Not required, as it is the default.

PORTRAIT

Displays the report across the narrower dimension of a vertical page, producing a page that is longer than it is wide. This value is the default.

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 Specify 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`

The TYPE specification is optional with this feature. If omitted, TYPE defaults to REPORT.

color

Is a supported color. For a list of supported colors, see *Color Values in a Report*.

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

Product Number#:	Store Name:	State:	Quantity In Stock:
1004	eMart	CA	689,088

Setting Page Margins

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 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 displays 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 ? STYLE Query

Use the ? 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

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* 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*.

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 the corresponding HTML font size:

Size in Points	Corresponding HTML Font Size
8 or smaller	1
9	2
10	3
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* 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

Category	Unit Sales	Dollar Sales
Coffee	*1376266*	*17231465*
Food	*1384845*	*17229344*
Gifts	*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* 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*.

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 *Identifying Report Components* 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*, or BACKCOLOR=*color*,

where *color* is one of the following values:

```
AQUA (CYAN)
AQUAMARINE
BLACK
BLUE VIOLET
CADET BLUE
CORAL
CORNFLOWER BLUE
CYAN (AQUA)
DARK GREEN
DARK OLIVE GREEN
DARK ORCHID
DARK SLATE BLUE (PURPLE)
DARK SLATE GREY
DARK TURQUOISE
DIM GREY (GRAY, GREY)
FIREBRICK
FOREST GREEN (GREEN)
FUCHSIA (MAGENTA)
GOLD
GOLDENROD
GRAY (DIM GREY, GREY)
GREEN (FOREST GREEN)
GREEN YELLOW
GREY (DIM GREY, GRAY)
INDIAN RED
KHAKI
LIGHT BLUE
LIGHT GREY
LIGHT STEEL BLUE
LIME
LIME GREEN
MAGENTA (FUCHSIA)
MAROON
MEDIUM AQUAMARINE
MEDIUM BLUE
MEDIUM FOREST GREEN (OLIVE)
MEDIUM GOLDENROD
MEDIUM ORCHID
MEDIUM SLATE BLUE
MEDIUM SPRING GREEN
MEDIUM TURQUOISE
MEDIUM VIOLET RED
MIDNIGHT BLUE
NAVY (NAVY BLUE)
OLIVE (MEDIUM FOREST GREEN)
ORANGE
ORANGE RED
ORCHID
PALE GREEN
PINK
PLUM
PURPLE (DARK SLATE BLUE)
RED
SALMON
SEA GREEN
SIENNA
SILVER
SKY BLUE
SLATE BLUE
STEEL BLUE (TEAL)
TAN
TEAL (STEEL BLUE)
THISTLE
TURQUOISE
VIOLET
VIOLET RED
WHEAT
WHITE
YELLOW
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

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* 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

Category	Product	Unit Sales	Dollar Sales
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

Reusing FOCUS StyleSheet Declarations With Macros

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

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

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. For an example, see *Defining, Applying, and Overriding a FOCUS StyleSheet Macro*.

Syntax How to Apply a FOCUS StyleSheet Macro

`TYPE=type, [subtype,] MACRO=macroname, [condition,] $`

where:

type

Is the value of the TYPE attribute. You can specify any report component.

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* for information about how to specify different types of report components.

type

Is the report component you wish to affect.

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

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

Sales Report - Page 1

Identifying Report Components

The basic concept behind StyleSheets is that a report consists of several components, each of which has a specific name. A StyleSheet file consists of style declarations for those components whose styles you want to change, along with the formatting that you want to apply to those components. Any component that you do not specifically format in your StyleSheet either retains the default style or inherits a style from a higher-level component. Inheritance is discussed in *FOCUS StyleSheet Attribute Inheritance*.

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*.
PAGENUM	Default page numbers. See *Identifying a Page Number, Underline, or Skipped Line*. **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*.
TABFOOTING	A footing on or after the last page of a report, generated by ON TABLE SUBFOOT. See *Identifying a Heading or Footing*.
HEADING	Headings at the top of each report page. See *Identifying a Heading or Footing*.
FOOTING	Footings at the bottom of each report page. See *Identifying a Heading or Footing*.
SUBHEAD	Headings before a particular sort field, generated by ON sortfield SUBHEAD. See *Identifying a Heading or Footing*.

TYPE	Report Component
SUBFOOT	Footings after a particular sort field, generated by ON sortfield SUBFOOT. See *Identifying a Heading or Footing*.
DATA	Report data. See *Identifying Data*.
TITLE	Column titles. See *Identifying a Column or Row Title*.
ACROSSTITLE	ACROSS field names (that is, field names used in ACROSS phrases). See *Identifying a Heading, Footing, Title, or FML Free Text*
ACROSSVALUE	ACROSS field values (that is, values of the ACROSS field). These values become column titles in the report. See *Identifying Data*.
SUBTOTAL	Totals generated by SUBTOTAL, SUB-TOTAL, RECOMPUTE, and SUMMARIZE. See *Identifying Totals and Subtotals*.
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*.
RECAP	Lines generated by ON field name RECAP or ON field name COMPUTE. See *Identifying Totals and Subtotals*.
UNDERLINE	Underlines generated by ON field name UNDER-LINE. See *Identifying a Page Number, Underline, or Skipped Line*.

TYPE	Report Component
SKIPLINE	Skipped lines generated by ON field name SKIP-LINE. See *Identifying a Page Number, Underline, or Skipped Line*.
FREETEXT	FML free text. See *Identifying a Heading, Footing, Title, or FML Free Text*.

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

You can apply formatting to an:

- **Entire report**. For more information, see *How to Identify the Entire Report*.
- **Entire column** within a report, both its title and data (including ROW-TOTAL columns). For more information, see *How to Identify an Entire Column*.

- **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*, and *How to Identify an Entire Total or Subtotal Row*.

  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*.

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
```

TYPE=REPORT

Cost Analysis

YEAR

Plant	2000 Line Total	2000 Line Cost of Goods Sold	2001 Line Total	2001 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

TYPE=REPORT,COLUMN=B1

TYPE=REPORT,ACROSSCOLUMN=N2

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 *Identifying a Column of Data*.
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
`Cn`	Identifies a display column by its position in the report. To determine the value of *n*, count only display fields from left to right, including NOPRINT fields. Do not count vertical sort (BY) fields or ROW-TOTAL fields. To select all display fields use C*.
`Bn`	Identifies a vertical sort (BY) column by its position in the report. To determine the value of *n*, count only vertical sort (BY) fields, including NOPRINTs, from left to right. To select all BY fields use B*.
`field`	Identifies a column by its field name. When a field occurs more than once, use field(*n*) to select a particular occurrence or field(*) to select all occurrences of the field.
`ROWTOTAL`	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(*n*) to select a particular total column. Use ROWTOTAL(*field*) to select the row total column for a particular field. Use ROWTOTAL(*) to select all row total columns in the report.

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 fieldname (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

Product Name:	Quantity In Stock:
AR2 35MM Camera 8 X	11499
AR3 35MM Camera 10 X	12444
Combo Player - 4 Hd VCR + DVD	13527
QX Portable CD Player	22000
ZC Digital PDA - Standard	33000
ZT Digital PDA - Commercial	21000
2 Hd VCR LCD Menu	43068
250 8MM Camcorder 40 X	60073
330DX Digital Camera 1024K P	12707
750SL Digital Camcorder 300 X	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 are formatted for every instance of the column in the report output. 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	*Line Total*	**Line Cost of Goods Sold**	*Line Total*	**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={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 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:

	AMOUNT
CASH ON HAND	*8,784*
DEMAND DEPOSITS	*4,494*
TIME DEPOSITS	*7,961*
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:

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 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*.

```
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:

PRODUCT 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**

Identifying 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*.
- **Columns of data**, including sort columns and display columns. For more information, see *How to Identify a Column of Data*.

- **Sort rows** (that is, ACROSS field values). For more information, see *How to Identify a Row of Horizontal Sort (ACROSS) Data*.
- **Totals and subtotals**. For more information, see *Identifying Totals and Subtotals*.

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
```

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 PLANTLNG 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

PLANT	QUARTER 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*.

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 N*n*, 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:

Product Name:	Quantity:	Line Total	Line Cost Of Goods Sold
DVD Upgrade Unit for Cent. VCR	121	$18,177.77	16,819.00
QX Portable CD Player	266	$33,847.80	26,334.00
110 VHS-C Camcorder 20 X	266	$70,901.90	66,234.00
120 VHS-C Camcorder 40 X	266	$76,511.09	68,894.00
150 8MM Camcorder 20 X	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.

`n`

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

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*.

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 request 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 only need to include the BY identifier 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 to apply formatting to. 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*.

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. The relevant StyleSheet declarations are highlighted in the request.

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.

```
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:

Plant	Order Num	Product Name:	Quantity:	Line Cost of Goods Sold
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***

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 is more than one RECAP or COMPUTE field 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:

Plant	Order Number:	Quantity:	Line Total	Line Cost of Goods Sold
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***

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 you specify in an EXL2K report.

Identifying a Column or Row Title

Within a StyleSheet you can identify a report's column titles and horizontal sort (ACROSS) values in order to format them. The following example illustrates where column titles and horizontal sort values are 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

            DEPARTMENT ◄──────────────────────────────────────── TYPE = ACROSSTITLE
            MIS                                 PRODUCTION ◄──────────── TYPE = ACROSSVALUE
  PAY_DATE            GROSS          DED_AMT            GROSS          DED_AMT ◄── TYPE = TITLE
  ---------------------------------------------------------------------------
  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*.

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*.

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:

n

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 displayed in .

```
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:

	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

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 is128. 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 <, >, &, etc. should not be used since 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 *NF888: Excel 2000* and *NF989: Excel 97*.

Identifying a Heading or Footing

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 also 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 said “Page 1 of 5”, you can use the <TABLASTPAGE system variable in conjunction with the <TABPAGENO system variable to do so.

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 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
```

```
PAGE     1

 CONFIDENTIAL INFORMATION                                  ◄── TYPE = TABHEADING
 SWIFTY INFORMATION GROUP - EMPLOYEE LIST BY DEPARTMENT

        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, this enables you to 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*.

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****

Manufacturing Plant	YEAR 2000	2001	2002
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*.

line_#

Identifies a line by its position in the heading or footing. You only need to include the LINE attribute 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 displays 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, ITME=3, STYLE=BOLD, $
ENDSTYLE
END
```

For an example, see *Identifying a Text String in a Heading Using Spot Markers*.

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 Plant	Quantity:
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*.

line_#

Identifies a line by its position in the heading or footing. You only need to include the LINE attribute 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 Plant	Quantity:
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:

Salary	Percent Increase	Date of Increase	Employee ID
$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

Salary	Percent Increase	Date of Increase	Employee ID
$10,000.00	.10	82/01/01	071382660
			112847612
	.12	81/01/01	071382660

Page 2 of 5

Identifying a Page Number, Underline, or Skipped Line

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
```

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	YEAR	Quantity:
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

Product Name:	Quantity In Stock:
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

Reference Section Separation Features

Feature	Description	Applies to
`SKIP-LINE`	Adds a blank line.	HTML (requires GRID=OFF) PDF PS
`TYPE=SKIPLINE`	Formats a blank line.	PDF PS
`UNDER-LINE`	Underlines a sort group.	HTML (requires GRID=OFF) PDF PS
`TYPE=UNDERLINE`	Formats an underline.	PDF PS
`STYLE={+|-}UNDERLINE`	Adds an underline to a report component, or removes an underline from a report component other than a column title.	HTML PDF PS
`BAR AS '{-|=}'`	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. For a list of supported values, see *Color Values in a Report*.

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.

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 is:

Sales Report

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 or Remove a Report Component Underline

`TYPE=type, [subtype,] STYLE=[±|-]UNDERLINE, $`

where:

type

Is the report component. For valid values, see *Identifying Report Components*.

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). This value is the default.

-

Removes an underline from an inherited text style.

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 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
ACTION	JAWS	SPIELBERG S.
	ROBOCOP	VERHOVEN P.
	TOTAL RECALL	VERHOVEN P.
	TOP GUN	SCOTT T.
	RAMBO III	MCDONALD P.
CHILDREN	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.
	BAMBI	DISNEY W.
CLASSIC	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.

Positioning a Report Component

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*.

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	Sets absolute or relative starting position of a column. An absolute position is the distance from the left margin of the printed paper. A relative position is the distance from the default position. After the first column, the default position is the end of the preceding column.	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*.

+

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 amper (&) 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.

+ 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.

- 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 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. It is specified in the request that the PRODUCT_DESCRIPTION field display 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

Product	Ordered Units
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

Product	Ordered Units
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

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

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 is 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

Category	Product	Unit Sales	Dollar Sales
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

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
```

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 is a formatting technique that 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.

You can 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*.

identifier

Selects one or more columns using the COLUMN attribute described in *Identifying an Entire Report, Column, or Row*.

acrosscolumn

Selects the same column under every occurrence of an ACROSS sort field using the ACROSSCOLUMN attribute described in *Identifying an Entire Report, Column, or Row*.

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

Product	Ordered Units
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 *December 2001*

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

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
`SQUEEZE`	Sets column width.	HTML (requires FONT=DEFAULT-FIXED) PDF PS
`SET SPACES`	Sets number of spaces between columns. See the *Creating Reports manual*.	HTML (requires SET STYLEMODE=FIXED)
`SEQUENCE`	Sets column order.	PDF PS
`FOLD-LINE`	Reduces report width by stacking columns. See See the *Creating Reports manual*.	PDF PS
`OVER`	Stacks columns by placing them over one another. See See the *Creating Reports manual*.	HTML PDF PS
`IN {n|+n}`	Sets absolute or relative starting position of a column. See See the *Creating Reports manual*.	HTML (requires SET STYLEMODE=FIXED) PDF PS

Determining Column Width

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)*.

```
[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. This value is the default.

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:

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 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)*.

```
[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. This value is the default.

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.

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 displays ...
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 look different than how it will print.

```
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:

Product	Unit Sales
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

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

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 *NF888: Excel 2000*.

Syntax How to Control Wrapping of Report Data Using a StyleSheet Attribute

To suppress 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 etc. that is needed to identify the report component that you are formatting. See *Identifying Report Components* 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 that the column width can be set to. The value represents the measure specified with the UNITS parameter.

Syntax How to Control Wrapping of Report Data using a SET Command

To suppress 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.

Controlling Wrapping With Alternative Methods

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
`SET SQUEEZE=OFF`	Forces the browser to display the entire column width and suppresses the wrapping of column data.
`SET SQUEEZE=ON`	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.
`SET GRID=OFF`	Automatically suppresses the wrapping of data.
`SET GRID=ON`	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 attribute to OFF, use the following syntax.

```
SET SQUEEZE=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 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

Store Name:	Order Number:	Sales
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

Store Name:	Order Number:	Sales
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

Store Name:	Order Number:	Sales
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

Store Name:	Order Number:	Sales
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

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 etc. 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

Product	Product Code	Vendor Name	Unit Price
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

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, resulting 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*.

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

Category	Budget Units	Unit Sales	Budget Dollars	Dollar Sales
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 default), 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 pre-calculated; 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

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, resulting 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 STYLESHEET *
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					
MT		WY		Total by Gender	
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:

PROD CODE	UNIT SOLD	RETAIL PRICE	REV
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. Since numeric output is right justified by default, in this example 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:

DED CODE	DEPARTMENT	BANK ACCT	DED AMT
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

To align text and data in headings and footings based on factors other than left/right/center justification, consider the following descriptions before deciding which alignment method best suits your needs.

Alignment Method	Applies to ...	When to use...	Related Methods
1) StyleSheet Attributes: HEADALIGN COLSPAN JUSTIFY **Details:** See *Aligning a Heading or Footing Element in an HTML Report*.	HTML EXL2K	**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. **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*.	

Alignment Method	Applies to ...	When to use...	Related Methods
2) StyleSheet Attributes: WIDTH JUSTIFY **Details:** See *Aligning Content in a Multi-Line Heading or Footing*.	HTML PDF PS	**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. **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.	For an HTML or EXL2K report, you can align specific items with HEADALIGN options.

Alignment Method	Applies to ...	When to use...	Related Methods
2) StyleSheet Attributes (continued)		**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. You define the width of the decimal item, then you 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.	

Alignment Method	Applies to ...	When to use...	Related Methods
3) StyleSheet Attribute: POSITION **Details:** See *Positioning a Report Component*.	PDF PS HTML (limited)	**To set starting positions for headings or footings, or items within them:** Use POSITION syntax to specify absolute and relative starting positions. 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. **To align heading and footing items with columns:** Use POSITION syntax to align a heading item with a column position. For example, the syntax `TYPE=SUBHEAD, LINE=1, ITEM=3,POSITION=SALES, $` places ITEM 3 of the sort heading at the horizontal position where the column SALES is.	For a PDF report, you can accomplish most positioning with WIDTH and JUSTIFY. For an HTML report, you can align a heading item with a column by setting the HEADALIGN attribute to BODY.

Aligning 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 HEADLIGN=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 compare sample output, see *Comparing Output Generated With HEADALIGN Options*.

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*.

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*.

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

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*.

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 amper (&) 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.

```
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 body of the report.

```
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=TXTSEG, 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 DESCRIPTION AS 'CATALOG' SUBFOOT
"Evening Course"
"Course Description: <TEXTFLD"
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLESHEET *
TYPE = REPORT, GRID = OFF, $
TYPE = SUBFOOT, HEADALIGN = BODY, $
TYPE = SUBFOOT, LINE = 2, STYLE = BOLD, $
ENDSTYLE
END
```

The output is:

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.

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

The HEADALIGN and COLSPAN syntax described in *Aligning a Heading or Footing Element in an HTML Report* 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. For an illustration, see *Aligning Data and Text in a Multi-Line Heading or Footing*.

- Combine columnar formatting with line-by-line formatting. See *Combining Column and Line Formatting to Align Items in a Sort Heading*.
- Align decimal points when the data displayed has varying numbers of decimal places. See *Aligning Decimals in a Multi-Line Heading or Footing*.

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. For an illustration, see *Combining Column and Line Formatting to Align Items in a Sort Heading*.

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* for information about acceptable variations.

You can use either or both approaches for a single heading or footing. For an illustration, see *Combining Column and Line Formatting to Align Items in a Sort Heading*.

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 amper (&) 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*.

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*.

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:

Item1: **Text**	Item 2: **Data values**	Item 3: **Text**	Item 4: **Values with decimal places**
Plant	BOS DAL, and so on	Quantity	*n,nnn,nnn*
Order Date	2002/01/02	Cost of Goods	*nnn,nnn,nnn.ddddd*
Store Code	1003MD	Line Total	*$nnn,nnn,nnn.dd*

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* and *Aligning Decimals in a Multi-Line Heading or Footing*.

Note: To take advantage of this feature for an HTML report, you must 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

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	Face Value	Bond	Face Value
---	---	---	---
Galosh Ltd.	22375.5784596	Galosh Ltd.	22375.5784596
Mukluk Inc.	1212345.457	Mukluk Inc.	1212345.457
Overshoe Inc.	232.45484	Overshoe Inc.	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*.

ProcedureHow 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. Text and 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 *top*, 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.
3-5	Defines the content for the *bottom*, 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.

Line #	Description
7-11	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. **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 effect we want to achieve, 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. The formatting of the *bottom*, three-line section of the heading is controlled by the following specifications: 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. 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. 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.

Line #	Description
7-11 (cont)	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. 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. The common width and justification definitions enforce the proper alignment of each item.
12	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. 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 a lot of development time.

Line #	Description
13-18	Specifies line-by-line formatting for the *top*, 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. 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. Item 2 on each line refers to the text area; it has a defined width of 1 inch and the text is left justified. Item 3 on each line refers to the data values; it has a defined width of 2 inches and the data is left justified. The common width and justification definitions enforce the proper alignment of each item. 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.

FOCUS StyleSheet Attribute Inheritance

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

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

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 GGSALES
   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**

Adding Grids and Borders

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.	PDF PS
VGRID	Control vertical grid display and grid line density.	PDF PS

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 this 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*.

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=TABHEADING, STYLE=BOLD, JUSTIFY=CENTER, BORDER=HEAVY,
     BORDER-COLOR=RED, BORDER-STYLE=DOTTED, $
ENDSTYLE
END
```

The output is:

Sales Report
****CONFIDENTIAL****
December 2002

Category	Budget Units	Unit Sales	Budget Dollars	Dollar Sales
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

State	Number of Households	Avg. Size	Avg. Income
CA	11478067	3	44925
FL	5968392	2	34264
NY	6799434	3	42023

Conditionally Formatting in a StyleSheet

You can conditionally format report components pr 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

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).

For an example, see *Using Sequential Conditional Formatting*.

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

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 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 STYLESHEET *
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 PAYSCALE STATUS NOPRINT
BY ID_NUM
WHERE PLANT EQ 'BOS' AND PAYSCALE 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 ID#	Last Name	First Name	Pay Level
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 PAYSCALE
BY STATUS SKIP-LINE
WHERE PLANT EQ 'BOS' AND PAYSCALE 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+

Current Status	Last Name	First Name	Pay Level
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 PAYSCALE
BY STATUS SKIP-LINE
WHERE PLANT EQ 'BOS' AND PAYSCALE 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+

Current Status	Last Name	First Name	Pay Level
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

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 and 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					
ENGLAND			FRANCE		
RETAIL_COST	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, you can 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 because 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	.	.	.	.	.	.

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. This is because 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. This is because 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:

Category	PERIOD	Unit Sales	Dollar Sales	MOVAVE
Coffee	1	61666	801123	801,123.0
	2	54870	682340	741,731.5
	3	61608	765078	749,513.7
	4	57050	691274	712,897.3
	5	59229	720444	725,598.7
	6	58466	742457	718,058.3
	7	60771	747253	736,718.0
	8	54633	655896	715,202.0
	9	57829	730327	711,158.7
	10	57012	724412	703,545.0
	11	51110	620264	691,667.7
	12	58981	762328	702,334.7
	13	0	0	694,975.6
	14	0	0	719,879.4
	15	0	0	705,729.9

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:

CAR	SALES
ALFA ROMEO	30200
AUDI	7800
BMW	80390
DATSUN	43000
JAGUAR	12000
JENSEN	0
MASERATI	0
PEUGEOT	0
TOYOTA	35030
TRIUMPH	0

Adding an Image to a Report

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 displays. 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 VM or in a data set allocated to ddname GIF in MVS.

Reference Image Attributes

Attribute	Description
IMAGE	Adds an image.
IMAGEALIGN	Positions an image. This applies only to HTML reports.
POSITION	Positions an image.

Attribute	Description
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. The description also displays as a pop-up description when your mouse or cursor hovers over the image in the report output.
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 a PDF, PS, or HTML Report With Internal Cascading Style Sheet*.

```
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.

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*.

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*. Enclose *column* in parentheses.

This option enables you to add different images to a heading or footing, depending on the value of the field. See *Using a File Name in a Data Source Field in an HTML Report*.

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.

The description also displays as a pop-up description when your mouse or cursor hovers over the image in the report output.

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 apply:

- A base URL must end with a slash (/).
- 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	DORA 2 DOOR

Cars produced in JAPAN

DATSUN	B210 2 DOOR AUTO
TOYOTA	COROLLA 4 DOOR DIX 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). The description also displays as a pop-up description when your mouse or cursor hovers over the image in the report output.

```
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

Product	Unit Sales
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*.

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:

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 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*.

`file`

PDF or PS report:

Is the name of the image file. On VM, the file must be on the CMS search path. On MVS, 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*. 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 that, 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. See *Using a File Name in a Data Source Field in an HTML Report*.

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 on the UNITS attribute.

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 on the UNITS attribute.

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 on the UNITS attribute.

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 on the UNITS attribute.

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=TABHEADING, 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

Product	Unit Sales
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

Product	Unit Sales
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

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

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*.

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.

Category	Region	Unit Sales	Dollar Sales
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 displays in your browser.

Linking 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 need to 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

You can create conditions when linking from a report. For example, you may only be interested in displaying 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*.

Example Linking With Conditions

In this example, we only want to link 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:

Department	Total Current Salaries
MIS	$108,002.00
PRODUCTION	$114,282.00

Linking 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 /approot/ibinccen/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*.

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

EMP_ID	LAST_NAME
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

If you want to 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

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 displays. 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 an 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*.

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

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

Next page or previous page.

Store Code	Order Number	Ordered Units
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

Next page or previous page. go back

Store Code	Order Number	Ordered Units
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

Store Code	Order Number	Ordered Units
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

Store Code	Order Number	Ordered Units
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

You can print sheets of laser printer 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 on the UNITS attribute.	PDF PS
GUTTER	Sets the horizontal and vertical distance between each label, expressed in the unit of measurement on the UNITS attribute.	PDF PS
MATRIXORDER	Sets the order in which the labels are printed.	PDF PS
LABELPROMPT	Sets the position of the first label on the mailing label sheet.	PDF PS

ProcedureHow 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=(c r), 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 8½ 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                    DIANE JONES
160 LOMBARDO AVE.
APT 4C                          235 MURRAY HIL PKWY
FREEPORT NY 11520               RUTHERFORD NJ 07073

ROSEMARIE BLACKWOOD             JOHN MCCOY
MRS. P. JONES                   ASSOCIATED
3704 FARRAGUT RD.               2 PENN PLAZA
BROOKLYN NY 11210               NEW YORK NY 10001

BARBARA CROSS                   ROGER MCKNIGHT
APT 2G                          APT 4D
147-15 NORTHERN BLD             117 HARRISON AVE.
FLUSHING NY 11354               ROSELAND NJ 07068

MARY GREENSPAN                  ANTHONY ROMANS

13 LINDEN AVE.                  271 PRESIDENT ST.
JERSEY CITY NJ 07300            FREEPORT NY 11520

JOAN IRVING                     MARY SMITH
APT 2J                          ASSOCIATED
123 E 32 ST.                    2 PENN PLAZA
NEW YORK NY 10001               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

DEPARTMENT	LAST_NAME	CURR_SAL
MIS	SMITH	$13,200.00
	JONES	$18,480.00
	MCCOY	$18,480.00
	BLACKWOOD	$21,780.00
	GREENSPAN	$9,000.00
	CROSS	$27,062.00

PAGE 2

DEPARTMENT	LAST_NAME	CURR_SAL
PRODUCTION	STEVENS	$11,000.00
	SMITH	$9,500.00
	BANNING	$29,700.00
	IRVING	$26,862.00
	ROMANS	$21,120.00
	MCKNIGHT	$16,100.00

Working With PostScript and PDF Reports

FOCUS generates a PDF or PS document from scratch. In order to do that 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	MVS	CMS
StyleSheet files. You can create them with a text editor (see *Creating a StyleSheet*).	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	MVS	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.	Any member of the variable length PDS allocated to ddname PS or PDF.	File type PS or PDF, any file name

Note that if you add fonts, you also need PFA or PFB files for those fonts.

Adding PostScript Type 1 Fonts for PS and PDF Formats

Through a simple process, you can customize your environment to take advantage of these fonts.

- First, you copy the font files (AFM and PFB or PFA) to FOCUS.

- Next, you 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 completed, 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`

ProcedureHow 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 MVS (or into a PDS with the same DCB attributes that's concatenated to it). On 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 MVS, 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 MVS or the file name on 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 MVS 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 MVS, 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 MVS member names or VM file names. For example, if the Windows designation is `font=garamond_light`, MVS 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 *.

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 *.

The entry must be in upper case and may not exceed 8 characters.

For illustrations, see *FOCUS Font Map File Entries for PDF* and *FOCUS Font Map File Entries for PS*.

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 then 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 then 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
```

NF959: DATASET for a Segment in a Master File

In addition to having a DATASET attribute at the file level in a FOCUS Master File, you can add the DATASET attribute to a segment declaration to specify the physical file name for a LOCATION segment or a cross-referenced segment with field redefinitions. In addition, the DATASET attribute permits you to bypass the FOCUS search mechanism for default data source location. DATASET eliminates the need to allocate data sources using JCL, FILEDEF, DYNAM, and USE commands.

User allocation and system specific behavior is as follows:

Platform	User allocation command
CMS	FILEDEF
MVS	DYNAM ALLOC or TSO ALLOC

Note: The MODIFY FIND function is not supported with the DATASET attribute. To use FIND with a data source, you must manually allocate the data source.

DATASET Behavior in FOCUS Data Sources

The DATASET attribute can be used on the file or segment level of a FOCUS Master File. If a USE command or explicit allocation is issued for the file, the DATASET attribute will be ignored.

The segment with the DATASET attribute must be either a LOCATION segment or a cross-referenced segment. For cross-referenced segments:

- If field declarations are specified for the cross-referenced fields, the DATASET attribute is the only method for specifying a physical file because FOCUS will not read the cross-referenced Master File and, therefore, will not be able to pick up its DATASET attribute if one is specified
- If field declarations are not specified for the cross-referenced fields, it is better to place the DATASET attribute at the file level in the cross-referenced Master File as FOCUS will read and parse the cross-referenced Master File. In this case, specifying different DATASET values at the segment level in the host Master File and the file level of the cross-referenced Master File causes a conflict, resulting in a (FOC1998) message.

If DATASET is used in a Master File whose data source is managed by the FOCUS Database Server, the DATASET attribute is ignored on the server side because the FOCUS Database Server does not read Master Files for servicing table requests.

The DATASET attribute in the Master File has the lowest priority:

- A user's explicit allocation overrides DATASET attributes.
- The USE command for FOCUS data sources overrides DATASET attributes and explicit allocations.

Note: If a DATASET allocation is in effect, a CHECK FILE command must be issued in order to override it by an explicit allocation command. The CHECK FILE command will de-allocate the allocation created by DATASET.

Syntax How to Use the DATASET Attribute

For a LOCATION segment:

```
SEGNAME=segname, SEGTYPE=segtype, PARENT=parent, LOCATION=filename1,
                 DATASET='filename2 [ON sinkname]' ,$
```

For a cross-referenced segment:

```
SEGNAME=segname, SEGTYPE=segtype, PARENT=parent, [CRSEGNAME=crsegname,]
[CRKEY=crkey,] CRFILE=crfile, DATASET='filename2 [ON sinkname]',
  FIELD=...
```

where:

filename2

Is the platform-dependent physical name of the data source.

sinkname

Indicates that the data source is located on the FOCUS Database Server. This attribute is valid for FOCUS data sources.

On MVS, the syntax is:

```
{DATASET|DATA}='qualifier.qualifier ...'
```

or

```
{DATASET|DATA}='ddname ON sinkname'
```

On CMS, the syntax is:

```
{DATASET|DATA}='filename filetype filemode [ON sinkname]'
```

Example Allocating a FOCUS Segment Using the DATASET Attribute

On MVS:

```
FILE = ...
SEGNAME=BODY,SEGTYPE=S1,PARENT=CARREC,LOCATION=BODYSEG,
 DATASET='USER1.BODYSEG.FOCUS',
 FIELDNAME=BODYTYPE,TYPE,A12,$
 FIELDNAME=SEATS,SEAT,I3,$
 FIELDNAME=DEALER_COST,DCOST,D7,$
 FIELDNAME=RETAIL_COST,RCOST,D7,$
 FIELDNAME=SALES,UNITS,I6,$
```

On CMS:

```
FILE = ...
SEGNAME=BODY,SEGTYPE=S1,PARENT=CARREC,LOCATION=BODYSEG,
 DATASET='BODYSEG FOCUS A',
 FIELDNAME=BODYTYPE,TYPE,A12,$
 FIELDNAME=SEATS,SEAT,I3,$
 FIELDNAME=DEALER_COST,DCOST,D7,$
 FIELDNAME=RETAIL_COST,RCOST,D7,$
 FIELDNAME=SALES,UNITS,I6,$
```

On MVS with SU:

```
FILE = ...
SEGNAME=BODY,SEGTYPE=S1,PARENT=CARREC,LOCATION=BODYSEG,
 DATASET='BODYSEG ON MYSU',
 FIELDNAME=BODYTYPE,TYPE,A12,$
 FIELDNAME=SEATS,SEAT,I3,$
 FIELDNAME=DEALER_COST,DCOST,D7,$
 FIELDNAME=RETAIL_COST,RCOST,D7,$
 FIELDNAME=SALES,UNITS,I6,$
```

Reference Messages

`(FOC1876) LOCATION IS NOT SPECIFIED FOR THE SEGMENT WITH DATASET: segment-name`

DATASET specification on a segment/class level requires LOCATION to associate physical data with a logical name.

FOC1876 displays if a DATASET attribute appears on a SEGMENT level with no LOCATION or cross-referenced file involved.

`(FOC1877) LOCATION WITH DATASET; INCONSISTENT DATASET VALUE FOR: location-file`

Multiple segments in a structure can specify the same LOCATION name. These LOCATION attributes cannot point to multiple DATASET values. This also applies to cross-referenced segments pointing to the same cross-referenced file.

FOC1877 displays if multiple segments are stored in the same LOCATION, and DATASET specifications are in conflict.

`(FOC1998) CONFLICTING DATASET SPECIFICATIONS FOR cross-ref-file`

Different DATASET values are specified at the FILE level of a cross-referenced file, and at the SEGMENT level of the host file.

FOC1998 displays when the DATASET value at the FILE level of a cross-referenced file, and the DATASET value at the SEGMENT level in the host file do not match.

`(FOC1999) INCORRECT 'ON' FILE NAME IN DATASET FOR master (ddname-in-dataset)`

When DATASET on file or segment level is used to specify sink machine as a data source (<filename> ON <sinkname> syntax), the file name must match the MFD name for file-level datasets, and LOCATION name for segment-level datasets.

FOC1999 displays if the name specified with the “ON *sink*” syntax is not the Master File name for a file level DATASET, and not the LOCATION name for a segment level DATASET.

NF961: USERFCHK Setting

The USERFCHK setting controls the level of verification applied to DEFINE FUNCTION and Information-Builders-supplied function arguments. It does not affect verification of the number of parameters; the correct number must always be supplied.

USERFCHK is not supported from Maintain.

Functions typically expect parameters to be a specific type or have a length that depends on the value of another parameter. It is possible in some situations to enforce these rules by truncating the length of a parameter and, therefore, avoid generating an error at run-time.

The level of verification and possible conversion to a valid format performed depends on the specific function. The following two situations can usually be converted satisfactorily:

- If a numeric parameter specifies a maximum size for an alphanumeric parameter, but the alphanumeric string supplied is longer than the specified size, the string can be truncated.
- If a parameter supplied as a numeric literal specifies a value larger than the maximum size for a parameter, it can be reduced to the proper value.

Prerequisites for Function Parameter Verification

Parameter verification can be enabled only for DEFINE FUNCTIONs and functions supplied by Information Builders. If your site has a locally written function with the same name as an Information-Builders-supplied function, the USERFNS setting determines which function will be used:

```
SET USERFNS= {SYSTEM|LOCAL}
```

where:

SYSTEM

Gives precedence to functions supplied by Information Builders. SYSTEM is the default setting. This setting is required in order to enable parameter verification.

LOCAL

Gives precedence to locally written functions. Parameter verification is not performed with this setting in effect.

Syntax How to Control Function Parameter Verification

Issue the following command in FOCPARM, FOCPROF, on the command line, in a FOCEXEC, or in an ON TABLE command. Note that the USERFNS=SYSTEM setting must be in effect

```
SET USERFCHK = setting
```

where:

setting

Can be one of the following:

ON

Is the default value. Verifies parameters in requests, but does not verify parameters for functions used in Master File DEFINEs. If a parameter has an incorrect length, an attempt is made to fix the problem. If such a problem cannot be fixed, an error message is generated and the evaluation of the affected expression is terminated.

Note that if a parameter provided is the incorrect type, verification fails and processing terminates.

Because parameters are not verified for functions specified in a Master File, no errors are reported for those functions until the DEFINE field is used in a subsequent request when, if a problem occurs, the following message is generated:

```
(FOC003) THE FIELDNAME IS NOT RECOGNIZED
```

OFF

Does not verify parameters except in the following cases:

- If a parameter that is too long would overwrite the memory area in which the computational code is stored, the size is automatically reduced without issuing a message.
- If an alphanumeric parameter is too short, it is padded with blanks to the correct length.

FULL

Is the same as ON, but also verifies parameters for functions used in Master File DEFINEs.

Note that if a parameter provided is the incorrect type, verification fails and processing terminates.

ALERT

Verifies parameters in a request without halting execution when a problem is detected. It does not verify parameters for functions used in Master File DEFINEs. If a parameter has an incorrect length and an attempt is made to fix the problem behind the scenes, the problem is corrected with no message. If such a problem cannot be fixed, a warning message is generated. Execution then continues as though the setting were OFF.

Note that if a parameter provided is the incorrect type, verification fails and processing terminates.

Example Verifying Parameters With Correctable Errors

The following request uses SUBSTR to extract the substring that starts in position 6 and ends in position 14 of the TITLE field. The fifth argument specifies a substring length (500) that is too long (it should be no longer than 9):

```
SET USERFCHK = ON
TABLE FILE MOVIES
PRINT TITLE
COMPUTE
  NEWTITLE/A9  = SUBSTR(39, TITLE, 6  ,14, 500, NEWTITLE);
WHERE CATEGORY EQ 'CHILDREN'
END
```

When the request is executed with USERFCHK=ON or OFF, the incorrect length is corrected and the request continues processing:

```
TITLE                                     NEWTITLE
-----                                     --------
SMURFS, THE                               S, THE
SHAGGY DOG, THE                           Y DOG, TH
SCOOBY-DOO-A DOG IN THE RUFF              Y-DOO-A D
ALICE IN WONDERLAND                        IN WONDE
SESAME STREET-BEDTIME STORIES AND SONGS   E STREET-
ROMPER ROOM-ASK MISS MOLLY                R ROOM-AS
SLEEPING BEAUTY                           ING BEAUT
BAMBI
```

Example Verifying Parameters With Uncorrectable Errors

The following request has an incorrect data type in the last argument to SUBSTR. This parameter should specify an alphanumeric field or format for the extracted substring:

```
SET USERFCHK = ON
TABLE FILE MOVIES
PRINT TITLE
COMPUTE
  NEWTITLE/F9  = SUBSTR(39, TITLE, 6  ,14, 500, 'F9');
WHERE CATEGORY EQ 'CHILDREN'
END
```

- When the request is executed with USERFCHK=ON, a message is produced and the request terminates:

```
ERROR AT OR NEAR LINE      5  IN PROCEDURE USERFC3 FOCEXEC
(FOC279) NUMERIC ARGUMENTS IN PLACE WHERE ALPHA ARE CALLED FOR
(FOC009) INCOMPLETE REQUEST STATEMENT
UNKNOWN FOCUS COMMAND  WHERE
 BYPASSING TO END OF COMMAND
```

- When the request is executed with USERFCHK=OFF, no verification is done and no message is produced. The request executes and produces incorrect results. In some environments, this type of error may cause abnormal termination of the application:

```
DIRECTOR     TITLE                                  NEWTITLE
--------     -----                                  --------
             SMURFS, THE                           *********
BARTON C.    SHAGGY DOG, THE                       *********
             SCOOBY-DOO-A DOG IN THE RUFF          *********
GEROMINI     ALICE IN WONDERLAND                          1
             SESAME STREET-BEDTIME STORIES AND SONGS  -265774
             ROMPER ROOM-ASK MISS MOLLY            *********
DISNEY W.    SLEEPING BEAUTY                       *********
DISNEY W.    BAMBI                                        0
```

Reference Warning Message for USERFCHK

The following message displays when there is a conflict between a parameter and the length specified in a user function.

(FOC36335) **PARAMETER LENGTH CONFLICT IN FUNCTION** *name* **ARG** *number*

The parameter length of a user function or a directive is either below or above enforced limits or conflicts with the length provided as a constant in another parameter.

NF964: Extensions to FORECAST

The FORECAST feature allows you to uncover trends in numeric data. Depending on the options you specify, it can also provide predicted values beyond the range of the values stored in the data source.

The following functionality has been added to FORECAST:

- The EXPAVE method now uses a bootstrapping technique for calculating predicted values beyond the range of the data. This means that the last real data value is used as a factor in every prediction.
- Double and triple exponential smoothing methods have been added.
- The SUM command used with the REGRESS method to create a new report column (non-recursive) calculates the regression based on the summed values.
- The FOCUS GRAPH facility is supported.

Reference Usage Notes for FORECAST

- If you use the ESTRECORDS parameter to enable the external sort to better estimate 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
```

Using Exponential Smoothing

The exponential moving average method (known as exponential smoothing) calculates a weighted average between the previously calculated value of the average and the next data point. The following enhancements have now been implemented:

- Double exponential smoothing (DOUBLEXP) has been added. This type of average better approximates data in which there is a trend.
- Triple exponential smoothing (SEASONAL) has been added. This type of average better approximates data in which there is a trend and a seasonal adjustment.
- When the data values are exhausted for single exponential smoothing (EXPAVE), the last real data value is used as the data value in every subsequent prediction. This has changed from the prior implementation in which the last average calculated was used as the next data value, making every predicted value a constant equal to the last calculated average.

Syntax How to Use Exponential Smoothing

The following syntax is for the EXPAVE method:

```
ON sfld RECAP fld1[/fmt] = FORECAST(fld2, interval, npredict,
'EXPAVE',npoint1);
```

The following syntax is for the DOUBLEXP method:

```
ON sfld RECAP fld1[/fmt] = FORECAST(fld2, interval, npredict,
'DOUBLEXP',npoint1, npoint2);
```

The following syntax is for the SEASONAL method:

```
ON sfld RECAP fld1[/fmt] = FORECAST(fld2, interval, npredict, 'SEASONAL',
nperiod, npoint1, npoint2, npoint3);
```

where:

sfld

Is the last ACROSS field in the request and must be a numeric or smart date field. If the request contains no ACROSS phrases, FORECAST works on the last BY field. However, FORECAST is only supported with ACROSS when the display command is SUM, WRITE, ADD, or COUNT.

fld1

Is a numeric field. It can be a real field, a virtual field, or a calculated field.

Note: The word FORECAST and the opening parenthesis must be on the same line as the syntax *fld1*=.

fmt

Is the display format for *fld1*. If it is omitted, the default format is D12.2. Even if *fld1* was previously reformatted using a DEFINE or COMPUTE command, the format specified in the RECAP command is respected.

`fld2`

Can be any numeric field. If it is the same as *fld1* (recursive), the predicted values will be appended to the report column after the data values, and the original column is not printed in the report. If it is a different name than *fld1* (non-recursive), this new column will be calculated containing both trend values (estimated values within the range of the existing data points) and, if you specify a non-zero number of predictions, forecast values (predictions beyond the range of the existing data points), while retaining the original field as a separate report column, if it was referenced in the display command. Note that *fld2* cannot be a date-time field or a numeric field with date display options.

`interval`

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

For a date sort field, 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.

`nperiod`

This parameter applies only to method SEASONAL. It is a positive whole number that specifies the number of data points per period.

npredict

Is the number of predictions for FORECAST to calculate. It must be a whole number greater than or equal to zero. Zero indicates that you want no predictions. Zero is only supported when the RECAP field is a new field (non-recursive), not the same field used as the first parameter to FORECAST. For the SEASONAL method, npredict is the number of *periods* to calculate. The number of *points* generated is:

```
nperiod * npredict
```

npoint1

Is a positive whole number used to calculate the weights for each term in the average. 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)
```

Using Single Exponential Smoothing

This method calculates an average that allows you to choose weights to apply to newer and older values.

The weight given to the newest value is *k*, where:

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

The quantity *n* is an integer greater than one. Increasing *n* increases the weight assigned to the earlier observations (or data instances) as compared to the later ones.

The next calculation of the exponential moving average (EMA) value is derived by the following formula:

```
EMA = (EMA * (1-k)) + (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 data value in every subsequent prediction for that sort group.

Example Calculating a New Exponential Moving Average Column

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 SMONTH/M = SDATE;
 PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
  SUM UNITS DOLLARS
  BY  CATEGORY BY PERIOD
  WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
  ON PERIOD RECAP EXPAVE/D10.1= FORECAST(DOLLARS,1,3,'EXPAVE',3);
END
```

The output is:

Category	PERIOD	Unit Sales	Dollar Sales	EXPAVE
Coffee	1	61666	801123	801,123.0
	2	54870	682340	741,731.5
	3	61608	765078	753,404.8
	4	57050	691274	722,339.4
	5	59229	720444	721,391.7
	6	58466	742457	731,924.3
	7	60771	747253	739,588.7
	8	54633	655896	697,742.3
	9	57829	730327	714,034.7
	10	57012	724412	719,223.3
	11	51110	620264	669,743.7
	12	58981	762328	716,035.8
	13	0	0	739,181.9
	14	0	0	750,755.0
	15	0	0	756,541.5
Food	1	54394	672727	672,727.0
	2	54894	699073	685,900.0
	3	52713	642802	664,351.0
	4	58026	718514	691,432.5
	5	53289	660740	676,086.3
	6	58742	734705	705,395.6
	7	60127	760586	732,990.8
	8	55622	695235	714,112.9
	9	55787	683140	698,626.5
	10	57340	713768	706,197.2
	11	57459	710139	708,168.1
	12	57290	705315	706,741.6
	13	0	0	706,028.3
	14	0	0	705,671.6
	15	0	0	705,493.3

Note:

- The number *n*, which is used to calculate the weights is 3.
- Three predicted values of EXPAVE are calculated within each value of CATEGORY. For values outside the range of the data, new PERIOD values are generated by adding the interval value (1) to the prior PERIOD value.
- There are no UNITS or DOLLARS values for the generated PERIOD values.
- Each average is computed using DOLLARS values where they exist. For predicted values beyond those points, the calculated EXPAVE values are used as new data points in the exponential average calculation.

  The first EXPAVE value (801,123.0) is the same as the first DOLLARS value.

  The second EXPAVE value (741,731.5) is calculated as follows. Note that because of rounding and the number of decimal places used, the value derived in this sample calculation varies slightly from the one displayed in the report output:

```
n=3 (number used to calculate weights)

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

EXPAVE = (EXPAVE*(1-k))+(new-DOLLARS*k) = (801123*0.5) +
(682340*0.50)
       =  400561.5 + 341170 = 741731.5
```

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

```
EXPAVE = (EXPAVE*(1-k))+(new-DOLLARS*k) =
(741731.5*0.5)+(765078*0.50)
       =  370865.75 + 382539 = 753404.75
```

The predicted EXPAVE values (starting with 739,181.9) are calculated using the same formula as used to calculate the trend values. However, the data point used to calculate each predicted value is the last actual data value (762328). For example, for period 13, the previous average is 716,035.8, therefore, the average is calculated as follows:

```
(716,035.8 * 0.5) + (762328 * 0.5) = 734534.8
```

```
EXPAVE = (EXPAVE * (1-k)) + (new-DOLLARS * k) = (716035.8*0.5) +
         (762328*0.50) = 358017.9 + 381164 = 739181.9
```

Using Double Exponential Smoothing

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

The smoothed value for the current time period 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:

```
DOUBLEXP(t) = k * datavalue(t) + (1-k) * ((DOUBLEXP(t-1) + b(t-1))
```

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

```
b(t) = g * (DOUBLEXP(t)-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:

```
forecast(t+m) = DOUBLEXP(t) + m * b(t)
```

where *m* is the number of periods ahead for the forecast.

Example Calculating a Double Exponential Smoothing Column

The following request sums the ACTUAL_YTD field of the CENTSTMT data source by period and calculates a single and double exponential moving average. The double exponential smoothing method better estimates the trend in the data points than the single smoothing method:

```
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:

```
                YTD
PERIOD          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

Seasonality (also called periodicity) is a tendency of the data to repeat itself in intervals over time. Triple exponential smoothing produces an exponential moving average that takes both the trend and seasonality into account by using three equations with three constants. This is called the Holt-Winters (HW) method. An example of the type of data that may have both trend and seasonality is sales data that is growing and in which 25% of sales always occur during December.

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

The smoothed value for the current time period 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 calculated trend value, b(*t*), is a weighted average of the difference between the current and previous average and the trend for the previous time period. Therefore, 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 calculated seasonal index, I(*t*), is a weighted average of the current data point divided by the current average and the seasonal index for the previous season. Therefore, 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 smoothed average. The first smoothed average is set to the first data value. Initial values for the periodicity 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.

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 + ... + (y(2L) - y(L))/L)
```

The calculation of the initial periodicity factor is based on the average of the data values within each period, A(j) (1<=j<=N):

```
A(j) = ( y((j-1)L+1) + y((j-1)L+2) + ... + y(jL) ) / L
```

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<= n <= L):

```
I(n) = ( y(n)/A(1) + y(L+n)/A(2) + ... + 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. You may need to use nonlinear optimization techniques to find the optimal constants.

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 example, the data has periodicity 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

This method estimates values by assuming that the dependent variable (y, the new calculated values) and the independent variable (x, the sort field values) are related by the following function, which represents a straight line:

$y = mx + b$

The value of *m* represents the slope of the line, and *b* represents the y-intercept.

Example Calculating a New Linear Regression Column

The SUM command now produces a regression column based on the summed values:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 SMONTH/M = SDATE;
 PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
  SUM UNITS DOLLARS
  BY              PERIOD
  WHERE SYEAR EQ 97
  ON PERIOD RECAP REGRESS/D10.1 = FORECAST(
                DOLLARS,1,3,'REGRESS');
END
```

The output is:

PERIOD	Unit Sales	Dollar Sales	REGRESS
1	147404	1864129	1,862,233.4
2	147838	1861639	1,868,423.6
3	151222	1874439	1,874,613.7
4	148864	1829838	1,880,803.9
5	153618	1899494	1,886,994.0
6	153854	1932630	1,893,184.2
7	161004	2005402	1,899,374.3
8	149099	1838863	1,905,564.5
9	152668	1893954	1,911,754.6
10	153522	1933705	1,917,944.8
11	148222	1814803	1,924,134.9
12	157881	2006455	1,930,325.1
13	0	0	1,936,515.2
14	0	0	1,942,705.4
15	0	0	1,948,895.5

Graphing a FORECAST Request

The GRAPH facility is supported with FORECAST.

Example Graphing a FORECAST Request With FOCUS

```
SET HISTOGRAM = OFF
GRAPH FILE CENTSTMT
SUM ACTUAL_YTD
  ACROSS 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:

```
 Y  YTD,Actual
 E  EXP
 D  DOUBLEXP

100,000,000.0 +
              I
              I                                          . . *
              I                                . . * . . . . E
 50,000,000.0 +                        . . * . . . . E . .
              I               . . * . . . . . E . .
              I     . . * . . . . E . .
              * . .
           .0 +----+----+----+----+----+----+----+----+----+----+

            2001/12             2002/02             2002/04
                     2002/01             2002/03             2002/05

                                  PERIOD
```

NF965: Raised Limit for EXTERNAL INDEX

The number of partitions that can be included in the creation of an external index has been increased from 120 to whatever number is available in the File Control Table (FCT), up to a maximum of 240. The File Control Table contains an entry for every FOCUS, Fusion, external index, and MDI file used in your FOCUS session.

Reference How to Create an External Index

In order to create an external index you must use the REBUILD EXTERNAL INDEX facility and then issue the appropriate USE command to associate the external index with your database.

Example Creating an External Index

In this example there are 122 concatenated partitions:

```
USE
ORDERS1 AS ORDERS
ORDERS2 AS ORDERS
.
.
.
ORDERS119 AS ORDERS
ORDERS120 AS ORDERS
ORDERS121 AS ORDERS
ORDERS122 AS ORDERS
END
```

The new external index is built as follows:

```
REBUILD
EXTERNAL INDEX
NEW
ORDIDX
ORDERS
L_SHIPDATE
NO
NO
```

In the preceding example the REBUILD process builds an external index on L_SHIPDATE and pulls the data from all of the partitions specified for the ORDERS database.

Reference Usage Notes for EXTERNAL INDEX

- The limit for all open files in a session is 255.
- Since an external index is a b-tree index, it can contain only one field.
- The external index file is limited to two GB in size.

Reference Messages for EXTERNAL INDEX

The following error message is issued when the number of partitions exceeds 240:

(FOC994) **EXTERNAL INDEX ERROR: MORE THAN %1 COMPONENTS**

There is a limit to the number of partitions that may be indexed.

The following error message is issued when partitions of data change after the external index is built and the index is explicitly stated in the TABLE request.

(FOC973) EXTERNAL INDEX INVALID CONFIGURATION:

Not all components used to build concatenated external index are in use at this time, or use directory contains the extra ones. It is not safe to use this index.

If AUTOINDEX is ON no error will result.

NF968: FMLINFO Function

FMLINFO is a function in the Financial Modeling Language (FML) that returns the FOR value associated with each row in an FML report. With this function, you can use the appropriate FOR value in a COMPUTE command to do drill downs and sign changes for each row in the report, even when the row is a summary row created using an OR list or an FML Hierarchy ADD command.

Syntax How to Retain FOR Values in an FML Request

```
fieldname/fmt = FMLINFO('FORVALUE',outfield); [NOPRINT]
```

where:

fieldname

Is the field to which the value is returned.

'FORVALUE'

Returns the FOR value associated with each row in an FML report. If the FML row was generated as a sum of data records using the OR phrase, FMLINFO returns the first FOR value specified in the list of values. If the OR phrase was generated by an FML Hierarchy ADD command, FMLINFO returns the FOR value associated with the parent specified in the ADD command.

outfield

Is either the name of the field to which the value is returned or the format of the output value enclosed in single quotation marks.

NOPRINT

Does not print the field in the report output.

Note: The setting FORMULTIPLE=ON enables an incoming record to be used on more than one line in an FML report. This setting is required in order to use the FMLINFO function.

Example Retrieving FOR Values for FML Hierarchy Rows

The following request creates a field called PRINT_AMT that is the negative of the NAT_AMOUNT field for account numbers less than 2500 in the CENTSYSF data source. The CENTGL data source contains the hierarchy information for CENTSYSF. Therefore, CENTGL is joined to CENTSYSF for the request:

```
SET FORMULTIPLE = ON
JOIN SYS_ACCOUNT IN CENTGL TO ALL SYS_ACCOUNT IN CENTSYSF
TABLE FILE CENTGL
SUM NAT_AMOUNT/D10 IN 30
COMPUTE PRINT_AMT/D10 = IF FMLINFO('FORVALUE','A7') LT '2500'
                        THEN -NAT_AMOUNT ELSE NAT_AMOUNT;

COMPUTE FORV/A4 = FMLINFO('FORVALUE', 'A4');

COMPUTE ACTION/A9  = IF FORV LT '2500'
                     THEN 'CHANGED' ELSE 'UNCHANGED';

FOR GL_ACCOUNT
2000 WITH CHILDREN 2 ADD AS CAPTION
END
```

Note that the parent value specified in the WITH CHILDREN ADD command (2000) is returned for the first row on the report. Each subsequent row is also a consolidated subsection of the hierarchy with a parent value that is returned by FMLINFO:

```
                                    Month
                                    Actual      PRINT_AMT  FORV  ACTION
                                    ------      ---------  ----  ------
Gross Margin                   -25,639,223     25,639,223  2000  CHANGED
  Sales Revenue                -62,362,490     62,362,490  2100  CHANGED
    Retail Sales               -49,355,184     49,355,184  2200  CHANGED
    Mail Order Sales            -6,899,416      6,899,416  2300  CHANGED
    Internet Sales              -6,107,890      6,107,890  2400  CHANGED
  Cost Of Goods Sold            36,723,267     36,723,267  2500  UNCHANGED
    Variable Material Costs     27,438,625     27,438,625  2600  UNCHANGED
    Direct Labor                 6,176,900      6,176,900  2700  UNCHANGED
    Fixed Costs                  3,107,742      3,107,742  2800  UNCHANGED
```

Example Using FMLINFO With an OR Phrase

In the following request, the FOR value printed for the summary line is 1010. However, FMLINFO returns the first value specified on the OR list, 1030:

```
SET FORMULTIPLE = ON
TABLE FILE LEDGER
SUM AMOUNT
COMPUTE RETURNEDFOR/A8 = FMLINFO('FORVALUE','A8');
FOR ACCOUNT
1010                  OVER
1020                  OVER
1030                  OVER
BAR                   OVER
1030 OR 1020 OR 1010
END
```

The output is:

```
      AMOUNT  RETURNEDFOR
      ------  -----------
1010   8,784  1010
1020   4,494  1020
1030   7,961  1030
      ------  --------
1010  21,239  1030
```

NF970: SUFFIX = TAB

Tab-delimited data sources are sequential data sources in which field values are separated by tabs.

Tab-delimited data sources cannot participate in joins.

Accessing a SUFFIX=TAB Data Source

A Master File containing the attribute SUFFIX=TAB can be used to access tab-delimited sequential data sources in which:

- Character values are not enclosed in double quotation marks.
- Each record is completely contained on one line and terminated with the CRLF character combination.

Creating a Tab-Delimited Data Source

You can create a tab-delimited data source using the command HOLD FORMAT TAB. This saves the data values as a variable-length text file with fields separated by tabs. Character values are not enclosed in double quotation marks. Leading blanks are removed from numeric fields and trailing blanks are removed from character fields.

A Master File is created for format TAB when the command used to create the output file is HOLD. The SUFFIX in the generated Master File is TAB.

All values are stored in their actual formats. The following display options are propagated to the output file: leading zeros, zero suppression, and scientific notation. Master file formats generated are exactly as in FORMAT ALPHA.

Syntax How to Save Report Output in Tab-Delimited Format

`[ON TABLE] command [AS filename] FORMAT TAB`

where:

`ON TABLE`

Is the syntax for creating an output file within a TABLE request.

command

Can be one of the following:

HOLD creates a Master File. SAVE creates an output file with no accompanying Master File.

filename

Is the file name of the resulting output file on VM, or the ddname on OS/390. The FILETYPE on VM for the output file is TAB.

Reference Usage Notes for Tab-Delimited Output Files

- There is an 8K overhead for processing tab-delimited files using HOLD FORMAT TAB. Therefore, extract files created by HOLD FORMAT TAB are limited to 24K record lengths.
- You must be aware of the formats and lengths supported by the application for which you are creating the output file. Field lengths created by the HOLD or SAVE command are those supported as USAGE formats in a Master File. You must take into account any limits of the destination application.
- If you use field-based reformatting in the request that creates the output file, the field lengths in the output file are the lengths from the original Master File or DEFINE command.

- By default, if the request that creates the output file retrieves missing data described with MISSING=ON in the original Master File, the output file will contain a blank if the original field was alphanumeric, or a zero if the field was numeric. You can change this behavior by issuing the command SET NULL=ON.
- As with all HOLD files, column headings specified with an AS phrase are ignored unless you set ASNAMES to ON. When ASNAMES is set ON, the value from the AS phrase, up to but not including the comma, becomes the field name in the HOLD Master File.
- The SAVB command is not supported. The output file is a text file.
- The OVER phrase is not supported.
- Smart date fields and dates formatted as I or P fields with date format options are treated as numeric and are not enclosed in double quotation marks in the output file. Dates formatted as alphanumeric fields with date format options are treated as alphanumeric and enclosed in double quotation marks.
- Continental decimal notation (CDN=ON|SPACE|QUOTE) is supported.
- Text fields are not supported.
- Blank field names will show up as blank column titles. This may result in an error when attempting to use the file as input to some applications.

- A double-precision number expressed in scientific notation is stored with the character D before the exponent. For example, the command PRINT RETAIL_COST/D9.3E prints the values in the following form: 0.888D+04. Some applications may not understand this value, as they generally expect an E before the exponent. Printing the value as single precision floating point instead of double precision produces a value with an E before the exponent. For example, format F9.3E produces the value 0.888E+04.

Example Creating a Tab-Delimited File

```
TABLE FILE MOVIES
PRINT COPIES TITLE WHOLESALEPR AS 'WHOLESALE,PRICE'
BY CATEGORY
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
ON TABLE HOLD AS TAB1 FORMAT TAB
END
```

For this example, the tabs are indicated by the > character in the following illustration of the generated tab-delimited file:

```
ACTION>2>JAWS>10.99
ACTION>3>ROBOCOP>11.50
ACTION>4>TOTAL RECALL>11.99
ACTION>2>TOP GUN>9.99
ACTION>3>RAMBO III>10.99
CHILDREN>1>SMURFS, THE>10.00
CHILDREN>2>SHAGGY DOG, THE>29.99
CHILDREN>1>SCOOBY-DOO-A DOG IN THE RUFF>9.75
CHILDREN>2>ALICE IN WONDERLAND>12.50
CHILDREN>1>SESAME STREET-BEDTIME STORIES AND SONGS>7.65
CHILDREN>1>ROMPER ROOM-ASK MISS MOLLY>7.99
CHILDREN>1>SLEEPING BEAUTY>15.99
CHILDREN>3>BAMBI>12.00
```

The HOLD command generates the following Master File:

```
FILE=TAB1            ,SUFFIX=TAB
SEGNAME=TAB1     ,SEGTYPE=S01
FIELDNAME   =CATEGORY           ,E01         ,A8        ,A08       ,$
FIELDNAME   =COPIES             ,E02         ,I3        ,A03       ,$
FIELDNAME   =TITLE              ,E03         ,A39       ,A39       ,$
FIELDNAME   =WHOLESALEPR        ,E04         ,F6.2      ,A06       ,$
```

NF971: SET NULL=ON

The SET NULL=ON command enables you to create a variable-length comma or tab delimited HOLD file that differentiates between a missing value and a blank string or zero value.

The HOLD formats supported for SET NULL=ON are COM, COMT, TAB, and TABT. Missing values in a record are denoted by two consecutive delimiters. A record that starts with a missing value has a delimiter in the first position, and a record that ends with a missing value has a delimiter in the last position.

Syntax How to Propagate a Missing Value to a Delimited File

```
SET NULL = {ON|OFF}
ON TABLE SET NULL {ON|OFF}
```

where:

`ON`

Propagates missing values to a delimited HOLD file when the field has MISSING=ON in the Master File.

`OFF`

Propagates the value zero for a missing numeric value and blank ("") for a missing alphanumeric value to a delimited HOLD file.

Example Creating a Comma-Delimited File With Missing Values

```
SET NULL = ON
TABLE FILE SALES
PRINT RETURNS DAMAGED
BY CITY BY DATE
ON TABLE HOLD AS NULL1 FORMAT COMT
END
```

The NULL1 file contains the following records:

`"CITY","DATE","RETURNS","DAMAGED"`	
`"NEW YORK","1017",2,3`	
`"NEW YORK","1017",2,1`	
`"NEW YORK","1017",0,1`	DAMAGED=0
`"NEW YORK","1017",,6`	DAMAGED is missing
`"NEW YORK","1017",4,`	RETURNS is missing
`"NEW YORK","1017",0,0`	DAMAGED=0, RETURNS=0
`"NEW YORK","1017",3,2`	
`"NEW YORK","1017",4,7`	
`"NEW YORK","1017",,`	DAMAGED is missing, RETURNS is missing
`"NEW YORK","1017",4,2`	
`"NEWARK","1018",1,1`	
`"NEWARK","1019",1,0`	RETURNS=0

Reference Usage Notes for SET NULL = ON

- When SET NULL = ON, the HNODATA setting is ignored. (See *NF940: SET HNODATA*.)
- A missing date is indistinguishable from an empty date.

NF972: NEWPAGE

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.

Syntax How to Create a Sort Heading or Sort Footing on a New Page

```
{BY|ON} fieldname SUBHEAD [NEWPAGE]
{BY|ON} fieldname SUBFOOT [MULTILINES][NEWPAGE]
```

where:

fieldname

Is a sort field in the request.

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.

MULTILINES

Suppresses the sort footing when there is only one line of data for a sort field value. MULTI-LINES is a synonym for MULTILINES.

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 would print at the bottom of the page for the prior region's data.

NF973: 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.

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).

What Are 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.

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

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://websrv1/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'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
B141	*Hazelnut*	*Coffee Connection*
B142	*French Roast*	*European Specialities,*
B144	*Kona*	*Evelina Imports, Ltd*
F101	*Scone*	*Ridgewood Bakeries*
F102	*Biscotti*	*Delancey 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

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

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

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

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 for 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's row is assigned to the cell.
2. If there are no conditional declarations, or if none of the conditions is 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

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.

```
TABLE FILE VIDEOTRK
HEADING
"Customer List </1"

PRINT LASTNAME AS 'Last,Name'
FIRSTNAME AS 'First,Name'
BY LOWEST 5 CUSTID AS 'Member,ID'

ON TABLE SET STYLE *
TYPE=REPORT,
CSSURL=http://websrvl/CSS/reportstyles.css, $
TYPE=DATA, COLUMN=LASTNAME, CLASS=italCol, $
.
.
.
END
```

FOCUS StyleSheet (in report procedure)

reportstyles.css

```
.italCol {font-style:italic}
.
.
.
```

external Cascading Style Sheet

Customer List

Member ID	Last Name	First Name
0925	*CRUZ*	IVY
0944	*HANDLER*	EVAN
1118	*WILSON*	KELLY
1133	*KRAMER*	CHERYL
1237	*GOODMAN*	JOHN

report output

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's heading using the Cascading Style Sheet's rule for the headText class.

4. Format the report's column titles using the CSS's rule for the reportTitles class.
5. For each report row for which the product'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'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'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'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'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
F101	*Scone*	*13.00*
F102	*Biscotti*	*17.00*
F103	Croissant	28.00
G100	*Mug*	*26.00*
G104	Thermos	96.00
G110	Coffee Grinder	125.00
G121	Coffee Pot	140.00

Combining CSS Styling With Other Formatting Methods

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, however, 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 a 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

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
F101	*Scone*	*13.00*
F102	*Biscotti*	*17.00*
F103	Croissant	28.00
G100	*Mug*	*26.00*
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. t

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.

NF977: IN-RANGES-OF

When you sort a report using a numeric sort field, you can group the sort field values together into equal ranges. The IN-RANGES-OF clause prints the range of sort values that defines each group. This differs from the IN-GROUPS-OF clause, which prints the starting point of each group.

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—that is, 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.

Reference Usage Notes for IN-RANGES-OF

- The first sort field range starts from the lowest value.
- Each report request can contain up to a total of five IN-RANGES-OF and IN-GROUPS-OF phrases.

- The IN-RANGES-OF phrase can only be used once per sort field.
- No message is generated if you specify a range of zero, but the values displayed on the report are unpredictable.

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
--------                          ---------
     5000.00 -      9999.99       SMITH
                                  GREENSPAN
    10000.00 -     14999.99       STEVENS
                                  SMITH
    15000.00 -     19999.99       JONES
                                  MCCOY
                                  MCKNIGHT
    20000.00 -     24999.99       ROMANS
                                  BLACKWOOD
    25000.00 -     29999.99       BANNING
                                  IRVING
                                  CROSS
```

NF979: SET HOLDFORMAT

HOLD files are, by default, stored in binary format. This hinders the ability to FTP these files to a PC. Using the SET HOLDFORMAT command, you can create HOLD files in alphanumeric format, facilitating the FTP of HOLD files to a PC.

Syntax How to Hold Files in Alpha or Binary Format

```
SET HOLDFORMAT = {BINARY|ALPHA}
```

or

```
ON TABLE SET HOLDFORMAT {BINARY|ALPHA}
```

where:

`BINARY`

Saves the HOLD file in binary format. BINARY is the default value.

`ALPHA`

Saves the HOLD file in alphanumeric format.

Reference Error Messages

If HOLDFORMAT is set to something other than ALPHA or BINARY the following error message is displayed:

```
SET HOLDFORMAT=123
VALID VALUES FOR HOLDFORMAT:ALPHA/BINARY
```

If HOLDFORMAT is set to a length greater than six characters the following error message is displayed:

```
SET HOLDFORMAT=TESTING
(FOC209) THE DATA VALUE EXCEEDS ITS LENGTH SPECIFICATION: TESTING
```

NF980: SET BLANKINDENT

To clarify relationships within an FML hierarchy, the captions (titles) of values are indented at each level. You can use the BLANKINDENT parameter in an HTML, PDF, or PostScript report to specify the indentation between each level of 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 widen the indented column if it is not wide enough to accommodate the indented fields. This is no problem in an HTML report. However, 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. You can change the width of a column using the StyleSheet SQUEEZE attribute. You can 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.

You can also 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. For information, see *NF983: Indenting FML Reports*.

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 would normally display before the caption. For child levels in an FML hierarchy, it indents 0.125 units for each space that would normally display between this line and the line above it.

`OFF`

Turns off indentations for FML hierarchy captions in an HTML report. 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 would normally display before the caption. For child levels in an FML hierarchy, it indents *n* units for each space that would normally display 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. They generate the following message, and the request processes as if BLANKINDENT=OFF:

```
VALID VALUES ARE OFF, ON OR A POSITIVE NUMBER (IN CURRENT UNITS)
```

You can change the number of blank spaces before the parent line of a hierarchy or before any FML tag or RECAP line in an FML request. For information, see *NF983: Indenting FML Reports.*

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 AS BLANKON  FORMAT HTML
END
```

The output is:

	Parent
Total Operating Expenses	1000
Selling Expenses	3000
Advertising	3100
Promotional Expenses	3100
Joint Marketing	3100
Bonuses/Commisions	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 the 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 AS BLANKON  FORMAT HTML
END
```

The output is:

	Parent
Total Operating Expenses	1000
Selling Expenses	3000
Advertising	3100
Promotional Expenses	3100
Joint Marketing	3100
Bonuses/Commisions	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 Widening an Indented Column With the SQUEEZE Attribute

The following request widens the column with the indented captions by specifying a width using the SQUEEZE attribute:

```
SET PAGE-NUM = NOPAGE
SET BLANKINDENT = .3
SET FORMULTIPLE = ON
TABLE FILE CENTGL
PRINT GL_ACCOUNT_PARENT  GL_ACCOUNT_LEVEL
FOR GL_ACCOUNT
3000  AS CAPTION       OVER
3000 GET  CHILDREN 2   AS CAPTION
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, FONT=COURIER, SIZE=10,$
TYPE=REPORT, COLUMN=N1, SQUEEZE=4, $
ENDSTYLE
END
```

The output is:

	Parent	Lev
Total Operating Expenses	1000	2
Selling Expenses	3000	3
Advertising	3100	4
Promotional Expenses	3100	4
Joint Marketing	3100	4
Bonuses/Commisions	3100	4
General + Admin Expenses	3000	3
Salaries-Corporate	3200	4
Company Benefits	3200	4
Depreciation Expenses	3200	4
Gain/(Loss) Sale of Equipment	3200	4
Leasehold Expenses	3200	4
Interest Expenses	3200	4
Utilities	3200	4

Example Moving a Column With the POSITION Attribute

The following request moves the column after the indented captions by specifying a starting position using the POSITION attribute:

```
SET PAGE-NUM = NOPAGE
SET BLANKINDENT = .3
SET FORMULTIPLE = ON
TABLE FILE CENTGL
PRINT GL_ACCOUNT_PARENT  GL_ACCOUNT_LEVEL
FOR GL_ACCOUNT
3000  AS CAPTION                 OVER
3000 GET  CHILDREN 2           AS CAPTION
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF, FONT=COURIER, SIZE=10,$
TYPE=REPORT, COLUMN=N2, POSITION=5, $
ENDSTYLE
END
```

The output is:

	Parent	Lev
Total Operating Expenses	1000	2
Selling Expenses	3000	3
Advertising	3100	4
Promotional Expenses	3100	4
Joint Marketing	3100	4
Bonuses/Commisions	3100	4
General + Admin Expenses	3000	3
Salaries-Corporate	3200	4
Company Benefits	3200	4
Depreciation Expenses	3200	4
Gain/(Loss) Sale of Equipment	3200	4
Leasehold Expenses	3200	4
Interest Expenses	3200	4
Utilities	3200	4

NF981: Increased Number of Sort Headings and Footings

You can use up to 32 SUBFOOTs and 32 SUBHEADs in a report. The total length of all of your sort headings and footings can be up to 32 K.

Reference Usage Notes for Increased Number of Sort Headings and Footings

- The maximum number of characters allowed in any single heading is 32 K.
- The maximum limit of headings is 64.

Reference Error Messages

If any of the limits mentioned above are exceeded, the following error message is displayed:

(FOC022) **THE MAXIMUM SPACE FOR COLUMN TITLES HAS BEEN EXCEEDED**

Each displayed data field may have from 1 to 5 lines as a column title. The total area allotted for column titles has been exceeded.

NF982: SET BYDISPLAY

Within a vertical sort group, the sort field value displays only on the first line of the rows for its sort group. 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 may enable 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. 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

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

The output is:

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

NF983: Indenting FML Reports

You can specify the number of spaces to indent the tag value, label text, or caption text for an FML tag row, hierarchy, or RECAP row. 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 information on FML hierarchies, see *NF951: Defining and Displaying FML Hierarchies*.

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 will be 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 on the FOR command, 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.

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 RECAP field's USAGE format. It cannot exceed the column width. The default is the format of the column in which the calculated value will be displayed.

expression

Is the expression that describes how to calculate the RECAP field's value.

Example Indenting an FML Tag Line and 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 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                    1000
     Indented 5                 1000
       Hierarchy Indented 5     3000
         Hierarchy Indented 5   3100
         Hierarchy Indented 5   3100
         Hierarchy Indented 5   3100
         Hierarchy Indented 5   3100
       Hierarchy Indented 5     3000
         Hierarchy Indented 5   3200
         Hierarchy Indented 5   3200
         Hierarchy Indented 5   3200
         Hierarchy Indented 5   3200
         Hierarchy Indented 5   3200
         Hierarchy Indented 5   3200
         Hierarchy Indented 5   3200
```

Example Indenting FML RECAP Lines

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:

```
                                        Our     Quantity
                         Price:         Cost:   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
```

Reference Messages for FML INDENT

(FOC36367) INDENT REQUIRES A POSITIVE INTEGER VALUE.

When using INDENT, the value specified must be a positive integer value.

Reference Usage Notes for FML INDENT

For information on the following techniques, see NF980: SET BLANKINDENT.

- If you want to specify an indentation other than two spaces between the levels of an FML hierarchy, you can do so in an HTML, PDF, or PostScript report using the SET BLANKINDENT command.
- If you want to retain the indentations between the levels of an FML hierarchy in an HTML report, SET BLANKINDENT cannot be OFF.

NF984: IEDIT Command

The IEDIT command opens the system editor (XEDIT on CMS or ISPF EDIT on MVS) from within FOCUS. This feature enables you to edit and execute variable length files and those with record lengths longer than 80 bytes, which TED does not support.

In addition, IEDIT adds the following TED-like functionality to the editing session:

In addition, IEDIT adds the following TED-like functionality to the editing session:

- You can issue the RUN command with or without parameters to save and execute a FOCEXEC that is open in the editor.
- The editor will be positioned to the line number where FOCUS encounters an error while executing a FOCEXEC.
- The last FOCEXEC executed will open when no file name is specified when you issue the IEDIT command.

All system editor commands are valid and any editor environment you establish as your default should be in force. TED commands other than RUN are not valid.

Syntax How to Edit Files With IEDIT

On CMS:

```
IEDIT [filename [[filetype|FOCEXEC] [filemode|A]]]
[(any_valid_XEDIT_option
```

On MVS:

```
IEDIT [ddname|FOCEXEC]
IEDIT [ddname(member)]
IEDIT [member]
```

where:

ddname

Is the DDNAME of the file to edit. If no DDNAME is supplied, it defaults to FOCEXEC.

member

Is the member name of the file to edit in the PDS allocated to ddname. Note: If only one name is specified in the command and it is both a ddname and a member name (in the FOCEXEC library), the file allocated to the ddname will be edited.

For example, if the FOCEXEC library has a member named FEX1 and a sequential file is allocated to ddname FEX1, the following command edits the sequential file:

```
IEDIT FEX1
```

filename

Is the name of the file to edit. If omitted, defaults to the name of the last FOCEXEC that was executed.

filetype

Is the filetype of the file to edit. If no filetype is supplied, it defaults to FOCEXEC.

filemode

Is the filemode of the file to edit. If the file exists on any disk, the mode of this file is used. If no file is found, it a new file is opened on A.

Reference Usage Notes for IEDIT

The TED command and the IEDIT command are not available once you are in the editor.

Example Editing a File With IEDIT on MVS

To edit a Master File (allocated to DDNAME MASTER) named CENTORD:

```
IEDIT MASTER(CENTORD)
```

To edit a FOCEXEC named LOADORD:

```
IEDIT LOADORD
```

To edit the last FOCEXEC that was run:

```
IEDIT
```

Example Editing a File With IEDIT on CMS

To edit a Master File (FILETYPE MASTER) named CENTORD:

```
IEDIT CENTORD MASTER
```

To edit a FOCEXEC named LOADORD:

```
IEDIT LOADORD
```

To edit the last FOCEXEC that was run:

```
IEDIT
```

IEDIT Facilities on CMS

When making changes to a FOCEXEC or an EXEC, the RUN command can be issued from the XEDIT command line. This is similar to the RUN command in TED. RUN can be issued for FOCEXECs and EXEC files.

RUN performs the following operations:

- Saves the file.
- Executes a FOCEXEC and passes it any parameters specified.
- Executes a CMS EXEC with an extended plist.

Installing IEDIT on MVS

Two files are required for using IEDIT. They are distributed as members of the FOCCTL.DATA data set and must be copied to a library in the concatenation of data sets allocated to DDNAME ISPTLIB.

- FOCIESTL. This is a macro that enables the RUN command and determines where in the file the editor will be positioned.
- FOCIERUN. This is a macro that processes the RUN command.

Installing IEDIT on CMS

Four files are required for using IEDIT:

- CONFIG IEDIT. This is the IEDIT profile shipped with Information Builders defaults that executes when IEDIT is invoked. This file should not be modified.
- PROFILE IEDIT. This file is for user-define IEDIT customization. This is the only one of these IEDIT configuration files that users should edit. For example, you can add:

  ```
  SET AUTOSAVE 5
  ```

- RUN IEDIT. This is a macro that enables the RUN command and should not be modified.

- IEDIT CONFIG. This is a REXX EXEC that is the interface between FOCUS and IEDIT. This file should not be modified.

A user can customize the look and feel of the XEDIT environment by making a copy of PROFILE IEDIT on the A disk and making changes as needed. All other files with a filename or filetype of IEDIT should not be modified.

ProcedureHow to Specify the Width of the Editor

The width of the editor depends on several factors. If the file exists and no width is specified on the IEDIT command or as a GLOBALV parameter, the LRECL of the file is honored. You can specify a width greater than the width of the file in two ways:

- In the IEDIT command

  `IEDIT fn ( WIDTH nnnn [any_valid_XEDIT_option]`

- With the GLOBALV setting:

  `GLOBALV SELECT IEDIT {SET|SETP} WIDTH nnnn`

  You can issue the GLOBALV command from CMS or from FOCUS (prefaced by 'CMS'). IEDIT then uses this width as the default width unless it is changed. Using SET in the GLOBALV command causes the setting to remain in effect for the VM session. Using SETP makes the setting permanent and lasts across VM sessions.

These widths are only honored if the WIDTH specified is greater than the width of the existing file. For files that do not exist, the WIDTH in the IEDIT command or the WIDTH in the GLOBALV setting is honored. If no width is given, and the file does not exist, it is opened as Fixed 80.

You can change the attributes of a file by issuing commands in XEDIT. For example:

```
RECFM V
```

However, you can only change the LRECL within the maximum width set for IEDIT, by default 256.

Using IEDIT on CMS

Most XEDIT features are available when you access XEDIT using the IEDIT command.

RIGHTP, LEFTP, RUN, SPVH, and SPHV are some of the commands that work differently when you issue the IEDIT command and the TED command. Many of these differences are due to the XEDIT environment.

- SPLITV and SPV split the screen into 2 vertical screens.
- SPLITH and SPH split the screen into 2 horizontal screens.
- SPVH and SPHV both split the IEDIT screen into four virtual screens.
- PF1/PF13 use CMS HELP to display the IEDIT HELP screen. TED uses the TED EDITOR to display the TED help screen.

When you have opened a file using the IEDIT command, you can open another file using the XEDIT command from the command line. The RUN command is still available.

Using IEDIT on MVS

On MVS, the editor is the ISPF editor with the RUN command added.

When you issue the =X command to exit ISPF, you return to FOCUS.

If you split the screen during an editing session, the new session is not part of FOCUS and does not have the RUN command or other IEDIT features available.

NF985: DYNAM Support for New GDG Iterations

The DYNAM command now supports allocation of new iterations of a Generation Data Group (GDG) using relative index numbers. Existing iterations are those with index numbers less than or equal to zero. A new iteration has an index number of +1.

Using DYNAM With Relative GDG Numbers

The original definition of a GDG assigns it a group name and specifies how many generations will be maintained. Once the maximum number of generations has been reached, each new iteration replaces the oldest existing iteration. The data set name for each iteration is the group name appended with a qualifier that contains the iteration number.

For example, the following GDG, named USER1.HOLD.GDG, has three generations:

Data Set Name	Iteration Number	Index Number
USER1.HOLD.GDG.G0003V00	3 (Oldest)	-2
USER1.HOLD.GDG.G0004V00	4	-1
USER1.HOLD.GDG.G0005V00	5 (Current)	0
USER1.HOLD.GDG.G0006V00	6 (New)	+1

The current (that is, most recent existing) iteration always has the highest existing iteration number and corresponds to index number zero. The next newest corresponds to index number -1, and so on. The iteration being created has an index number of +1 and is assigned the next iteration number in its data set name.

Prior to this new feature, DYNAM could allocate only existing iterations of a GDG. Now DYNAM supports relative index numbers for allocating a new iteration.

Syntax How to Use DYNAM to Allocate a New Iteration of a GDG

Use the following syntax to allocate a new iteration of a GDG

```
DYNAM ALLOC FILE ddname DS 'group_name(index)' NEW CATLG -
REFDSN file_with_DCB_attributes_to_use
```

where:

ddname

Is the name of the Master File for the GDG.

group_name

Is the group name of the GDG.

index

Is the index number for a new iteration of the GDG (usually +1).

file_with_DCB_attributes_to_use

Is the name of a data set whose DCB attributes should be used in creating the new iteration.

Example Allocating a New GDG Iteration Using DYNAM With Relative Index Numbers

The GDG named USER1.HOLD.GDG discussed in *Using DYNAM With Relative GDG Numbers* is described by Master File GDG1:

```
FILENAME=GDG1,SUFFIX=FIX
SEGNAME=ORIGIN,SEGTYPE=S1
 FIELDNAME=FLD1,,A10,$
 FIELDNAME=FLD2,,A20,$
 FIELDNAME=FLD3,,A30,$
 FIELDNAME=FLD4,,A10,$
```

The following command allocates a new iteration (index number +1)

```
DYNAM ALLOC FILE GDG1 DS 'USER1.HOLD.GDG(+1)' NEW CATLG -
REFDSN EDAQA.CAR.FOCUS
```

NF986: GROUP Fields in FOCUS Data Sources

A group can provide a convenient alternate name for one or more contiguous fields. The group redefines a sequence of fields and does not require any storage of its own.

Group fields allow you to:

- Join to multiple fields in a cross-referenced data source. You redefine the separate fields as a group, index this group key, and join to the group field.
- Automatically use indexes in MODIFY when the root segment has multiple indexed fields that comprise the key. In this case you define the group as the key. The SEGTYPE then becomes S1 and MODIFY automatically uses the index.
- Use an indexed view in a TABLE request when records are selected based on an equality test on each component of the group. TABLE only uses one index for this type of retrieval, so you can index the group and enhance retrieval efficiency.

The component fields can contain alphanumeric or numeric data. However, the group field should always have an alphanumeric format. The length of the group field must be the sum of the actual lengths of the component fields. For example, Integer fields always have an actual length of four bytes regardless of the USAGE format that determines how many characters are displayed on a report.

Syntax How to Define a Group Field

```
GROUP=groupname, [ALIAS=groupalias,] USAGE=An, [,FIELDTYPE=I] $
 FIELDNAME=field1, ALIAS=alias1, USAGE=fmt1,$
 FIELDNAME=field2, ALIAS=alias2, USAGE=fmt2,$
 .
 .
 .
 FIELDNAME=fieldn, ALIAS=aliasn, USAGE=fmtn,$
```

where:

groupname

Is the name of the group.

groupalias

Is an optional alias name for the group.

A*n*

Is the format for the group field. Its length is the sum of the internal lengths of the component fields:

- Fields of type I have an internal length of 4.
- Fields of type F have an internal length of 4.
- Fields of type P that have a USAGE format of P15 or P16 or smaller have an internal length of 8, and fields of type P that have a USAGE format of P17 or greater have an internal length of 16.
- Fields of type D have an internal length of 8.
- Fields of type A have an internal length equal to the number of characters (*n*) in their USAGE formats.

Describing the group field with a format other than A does not generate an error message; however, it is not recommended and may produce unpredictable results.

field1, ..., fieldn

Are the component fields in the group.

alias1, ..., aliasn

Are the alias names for the component fields in the group.

fmt1, ..., fmtn

Are the USAGE formats of the component fields in the group.

FIELDTYPE=I

Creates a separate index for the group.

Reference Usage Notes for Group Fields in FOCUS Data Sources

- When at least one component of the group has a numeric or date format, the group field does not display correctly. However, the value of the group field is correct, and the individual fields display correctly. The main use for this type of group in a TABLE request is to use indexed retrieval on more than one component field or to join to more than one component field.
- You can add or remove a non-key group field definition in a Master File at any time without impact. If the group field is indexed and you MODIFY the data source without the group field definition in the Master File, you must rebuild the index when you add the group field back to the Master File.
- The MISSING attribute is not supported on a group field.

- To use a group field that contains a numeric component in screening criteria, separate the component values with slashes. However, if the value of one the group components contains a slash '/', slash may not be used as a delimiter and no test can be issued against this value.

  Note that use of slashes makes it easier to specify values when the component fields contain trailing blanks because you do not have to account for those blanks.
- The only masks supported in screening criteria for group fields are those that accept any combination of characters for all group components after the first component. For example, if the FULL_NAME group consists of LAST_NAME and FIRST_NAME, the following mask is supported:

  ```
  WHERE FULL_NAME EQ '$R$$$$*'
  ```
- To use a group field that contains only alphanumeric components in screening criteria, you do not have to separate the component values with a slash, but you can.
- A group format supports date display options.
- In MODIFY, you can match on the group field when it is the first field in the segment (S1). A MATCH on a component field in the group without matching on the group generates the following message:

  ```
  (FOC439) WARNING. A MATCH CONDITION HAS BEEN ASSUMED FOR:
  ```
- If the group has an index, and the group components are also indexes, you can MATCH on the group level with no need to match on the group components.
- Although MODIFY will allow you to update group components even if they are key fields, this is not recommended. The recommended method for updating key fields is by using SCAN or FSCAN.

- If a group is part of the key, and the field following a group is also part of the key, the number specified for the SEGTYPE attribute must include the group field plus its component fields plus the field following the group. For example, if the group key has two components and is followed by another field that is also part of the key, the SEGTYPE must be S4:

```
GROUP = ...,                ,$
  FIELDNAME = FIELDG1, ... ,$
  FIELDNAME = FIELDG2, ... ,$
FIELDNAME = FIELD3, ...    ,$
```

Example Displaying an Alphanumeric Group Field

In the following group field definition, the group length is 25, the sum of the lengths of the LAST_NAME and FIRST_NAME fields:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,        ALIAS=EID,      FORMAT=A9,        $
 GROUP=FULL_NAME,                  ,      FORMAT=A25,       $
  FIELDNAME=LAST_NAME,     ALIAS=LN,       FORMAT=A15,      $
  FIELDNAME=FIRST_NAME,    ALIAS=FN,       FORMAT=A10,      $
```

The WHERE test on the group field does not need slashes between the component values because both component fields are alphanumeric:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID HIRE_DATE
BY FULL_NAME
WHERE FULL_NAME GT 'CROSSBARBARA'
END
```

The output is:

```
FULL_NAME                    EMP_ID     HIRE_DATE
---------                    ------     ---------
GREENSPAN       MARY         543729165   82/04/01
IRVING          JOAN         123764317   82/01/04
JONES           DIANE        117593129   82/05/01
MCCOY           JOHN         219984371   81/07/01
MCKNIGHT        ROGER        451123478   82/02/02
ROMANS          ANTHONY      126724188   82/07/01
SMITH           MARY         112847612   81/07/01
SMITH           RICHARD      119265415   82/01/04
STEVENS         ALFRED       071382660   80/06/02
```

Example Screening on a Group Field With an Integer Component

In the following group field definition, the group length is 29, the sum of the lengths of the LAST_NAME, FIRST_NAME, and HIRE_DATE fields. Because HIRE_DATE is an Integer field, its internal length is 4:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,       ALIAS=EID,     FORMAT=A9,      $
 GROUP=FULL_NAME,                ,      FORMAT=A29,      $
  FIELDNAME=LAST_NAME,    ALIAS=LN,      FORMAT=A15,     $
  FIELDNAME=FIRST_NAME,   ALIAS=FN,      FORMAT=A10,     $
  FIELDNAME=HIRE_DATE,    ALIAS=HDT,     FORMAT=I6YMD,   $
```

In the following request, the component field values must be separated by slashes in the WHERE test. The request does not display the group field because the integer component would not display correctly:

```
TABLE FILE EMPGROUP
PRINT EMP_ID LAST_NAME FIRST_NAME HIRE_DATE
BY FULL_NAME NOPRINT
WHERE FULL_NAME GT 'CROSS/BARBARA/811102'
END
```

The output is:

```
EMP_ID     LAST_NAME        FIRST_NAME  HIRE_DATE
------     ---------        ----------  ---------
543729165  GREENSPAN        MARY         82/04/01
123764317  IRVING           JOAN         82/01/04
117593129  JONES            DIANE        82/05/01
219984371  MCCOY            JOHN         81/07/01
451123478  MCKNIGHT         ROGER        82/02/02
126724188  ROMANS           ANTHONY      82/07/01
112847612  SMITH            MARY         81/07/01
119265415  SMITH            RICHARD      82/01/04
071382660  STEVENS          ALFRED       80/06/02
```

NF987: MDI for FOCUS and XFOCUS Data Sources

Note: As FOCUS 7.3 goes production, this feature is still in the process of undergoing the extensive testing effort needed before certifying it. It is, therefore, deactivated in the initial release of FOCUS 7.3. It will be activated in a future service pack after it has completed the testing cycle.

A multi-dimensional index (MDI) enables you to efficiently and flexibly retrieve information you need for business analysis. It looks at data differently from transaction processing systems in which the goal is to retrieve records based on a key. (FOCUS uses a B-tree index for this type of retrieval). The MDI is for retrieval only. It is not used for MODIFY or Maintain requests.

Business analysts may be interested in specific facts (data values, also called measures) about multiple categories of data in the data source. Categories of data, such as region or department, are referred to as dimensions. A multi-dimensional index uses dimensions and all of their hierarchical relationships to point to specific facts.

The MDI is a multi-field index that contains at least two dimensions. This index behaves like a virtual cube of values that intersect at measures of interest. The more dimensions used in a query, the better the retrieval performance.

For example, suppose that the CENTORD data source has an MDI with dimensions STATE, REGION, and PRODCAT.

The MDI is used to retrieve the facts (LINEPRICE and QUANTITY data) that lie at the intersection of the dimension values specified in the following request:

```
TABLE FILE CENTORD
SUM QUANTITY LINEPRICE
WHERE REGION EQ 'EAST'
WHERE STATE EQ 'DC'
WHERE PRODCAT EQ 'Cameras'
END
```

The MDI also provides the following other retrieval enhancing features: MDI JOIN, Dimensional JOIN, MDI WITHIN, MAXVALUES, MDI Encoding, and AUTOINDEX for MDI.

Syntax How to Enable a Multi-Dimensional Index (MDI)

To create and use an MDI, issue the following command in the FOCPARM ERRORS file:

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

where:

ON
: Enables MDIs.

OFF
: Disables MDIs. OFF is the default value.

Specifying an MDI in the Access File

All MDI attributes are specified in the Access File for the data source. The only attribute needed in the Master File is the ACCESSFILE attribute to point to the Access File containing the MDI specifications.

You also need to turn on the MDI feature by issuing the SET MDIFEATURE=ON command in your FOCPARM ERRORS file.

An MDI can be partitioned into multiple MDI files. However, even if the data source on which the MDI is built is partitioned, each MDI partition spans all data source partitions.

Syntax How to Specify an MDI in an Access File

```
MASTERNAME masterfile
    DATA[NAME] database_filename1
      .
      .
      .
    DATA[NAME] database_filenamen

    MDILOCATION mdiname
        TARGET_OF segname
        {DIM[ENSION]|FIELD[NAME]} field1 [MAXVALUES n1] [WITHIN dimname1]
          .
          .
          .
        {DIM[ENSION]|FIELD[NAME]}fieldn [MAXVALUES nn] [WITHIN dimnamen]

        DATA[NAME] mdifile1
          .
          .
          .
        DATA[NAME] mdifilen
```

where:

masterfile

Is the Master File name. You can have multiple MASTERNAME attributes in one Access File.

database_filename1, ..., database_filenamen

Are fully qualified physical file names in the syntax native to your operating environment. If the name contains blanks or special characters, it must be enclosed in single quotation marks. Multiple DATANAMEs describe concatenated partitions.

mdiname

Is the logical name of the MDI.

segname

Is the segment that contains the facts pointed to by the MDI. If the target data is distributed among several segments, the target should be the top segment that contains MDI data in order to avoid the multiplicative effect.

field1, ..., fieldn

Are the fields to use as dimensions. At least two dimensions are required for an MDI.

mdifile1,..., mdifilen

Are fully qualified physical file names for the MDI in the syntax native to your operating environment. If the name contains blanks or special characters, it must be enclosed in single quotation marks. Multiple DATANAMEs describe concatenated partitions.

n1, ... ,nn

Is the number of distinct values the field can have. This number must be a positive integer.

dimname1, ... , dimnamen

Defines a hierarchy of dimensions. This dimension is defined within the *dimname* dimension. For example, CITY WITHIN STATE.

Example Defining an MDI on CMS

This example shows an MDI with two partitions:

```
MASTERNAME CAR
 DATANAME 'CAR FOCUS M'
 MDILOCATION CARMDI
   TARGET_OF ORIGIN
   DIMENSION CAR
   DIMENSION COUNTRY
   DIMENSION MODEL
   DATANAME 'CARMDI1 MDI M'
   DATANAME 'CARMDI2 MDI M'
```

Example Defining an MDI on MVS

This example shows an MDI with two partitions:

```
MASTERNAME CAR
 DATANAME USER1.CAR.FOCUS
 MDILOCATION CARMDI
   TARGET_OF ORIGIN
   DIMENSION CAR
   DIMENSION COUNTRY
   DIMENSION MODEL
   DATANAME USER1.CAR1.MDI
   DATANAME USER1.CAR2.MDI
```

Creating a Multi-Dimensional Index

Each MDI is specified in an Access File for the data source. FOCUS uses the Access File in its retrieval analysis for each TABLE request.

You then use the REBUILD MDINDEX command to build the MDI. The MDI has the following DCB attributes: RECFM=F,LRECL=4096,BLKSIZE=4096.

A multi-dimensional index gives complex queries high-speed access to combinations of dimensions across data sources. If you know what information users want to retrieve and why, you can make intelligent choices about index dimensions.

An Access File can define more than one MDI. If the Access File defines multiple MDIs, the AUTOINDEX facility chooses the best index to use for each query.

The first step in designing an MDI is to find out what kind of information users need from the data source. You can get advice about your MDIs directly from FOCUS.

Choosing Dimensions for Your Index

The choice of index dimensions depends on knowing the data and on analyzing what is needed from it. Examine the record selection (IF and WHERE) tests in your queries to see how many index dimensions each application actually uses to select records. If different applications need different subsets of dimensions, consider separate MDIs for the separate subsets. Although FOCUS can produce high-speed reporting performance with indexes of up to 30 dimensions, smaller indexes usually generate less retrieval overhead. You can create an unlimited number of MDIs.

The following are good candidates for dimensions in an MDI:

- Fields used frequently in record selection tests. Multiple fields used in selection tests within one query belong in the same MDI.
- Fields used as the basis for horizontal partitioning, such as date or region.
- Derived dimensions (DEFINE fields) that define a new category based on an existing category; for example, if your data source contains the field STATE but you need region for your analysis, you can use the STATE field to derive the REGION dimension.
- Fields with many unique values. Fields which have few possible values are not normally good candidates. However, you may want to index such a field if the data source contains very few instances of one of the values, and you want to find those few instances.
- Packed decimal fields may be used as MDI dimensions on all platforms.
- An MDI can include a field that has a B-tree index as a dimension.
- MDI dimensions support missing values.

Including a field that is updated frequently (such as an AUTODATE field) in the MDI, requires frequent rebuilding of the MDI in order to keep it current. FOCUS can advise you on selecting MDI dimensions.

DEFINE fields described in the Master File can be used as dimensions. Dynamic DEFINE fields cannot be dimensions.

An MDI is for retrieval only; FIND and LOOKUP are not supported on an MDI.

Reference Guidelines for a Multi-Dimensional Index

The following guidelines apply to each MDI:

- The maximum size of an MDI is 200 GB.
- The maximum size of each index partition is 2 GB.
- The total size of all dimensions in an MDI cannot exceed 256 bytes. However, if you include the MAXVALUES attribute in the Access File declaration for a dimension, FOCUS uses a small number of bytes to store the values of that dimension:

MAXVALUES	Number of Bytes Required
1 through 253	1
254 through 65,533	2
Greater than 65,533	4

To allow for expansion, if the maximum number of values is close to a limit, make MAXVALUES big enough to use a larger number of bytes. For example, if you have 250 values, specify 254 for MAXVALUES, and reserve 2 bytes for each dimension value.

Building and Maintaining a Multi-Dimensional Index

The REBUILD command is used to create or maintain a multi-dimensional index. This command can be issued in a FOCUS session or a FOCEXEC.

The best MDI is built by specifying the dimensions in order of best cardinality (most distinct values).

If issued in a FOCUS session, the REBUILD command conducts a dialogue with the user. To issue the REBUILD command in a FOCEXEC, you place the REBUILD command and the user-supplied portion of the REBUILD dialogue in the stored procedure.

If the MDI file might be larger than two gigabytes or if you plan to add more data partitions to it, the MDI index file must be partitioned from the initial REBUILD phase. Once the index has been created, you can use it in a retrieval request. You cannot use an MDI for modifying the data source. If you update the data source without rebuilding the MDI and then attempt to retrieve data with it, FOCUS displays a message indicating that the MDI is out of date. You must then rebuild the MDI.

Example Creating a Multi-Dimensional Index in a FOCUS Session

This example illustrates the creation of an MDI, CARMDI. The Access File contains the following MDI attributes

```
MASTERNAME CAR
 DATANAME 'CAR FOCUS A'
 MDILOCATION CARMDI
   TARGET_OF ORIGIN
   DIMENSION CAR
   DIMENSION COUNTRY
   DIMENSION MODEL
   DATANAME 'CARMDI MDI A'
```

The top segment in the CAR data source, segment ORIGIN, is the TARGET_OF segment for the MDI. The following is the REBUILD procedure for the creation of the MDI.

1. Issue REBUILD at the command prompt.

   The REBUILD command invokes the REBUILD utility.

   The following menu displays:

```
Enter option
1. REBUILD (Optimize the database structure)
2. REORG (Alter the database structure)
3. INDEX (Build/modify the database index)
4. EXTERNAL INDEX (Build/modify an external index database)
5. CHECK (Check the database structure)
6. TIMESTAMP (Change the database timestamp)
7. DATE NEW (Convert old date formats to smartdate formats)
8. MDINDEX (Build/modify a multidimensional index.)
9. MIGRATE (Convert FOCUS masters/DBs to FUSION)
```

2. Enter MDINDEX. The MDINDEX command invokes the MDI option of the REBUILD utility.

   The following message appears:

```
NEW/ADD
```

3. Enter NEW. The NEW/ADD option enables you to create a new MDI or to add new data partitions to an existing MDI.

   The following message appears:

```
ENTER THE NAME OF THE MASTER
```

4. Enter CAR. The CAR Master File contains the segment that is the TARGET_OF the MDI.

   The following message appears:

   ```
   ENTER MD_INDEX LOCATION FILE NAME
   ```

5. Enter CARMDI. CARMDI is the logical name of the CARMDI MDI.

   The following message appears:

   ```
   ANY RECORD SELECTION TESTS? (YES/NO)
   ```

6. Enter NO to indicate that there are no record selection criteria for the index. The user can limit the index to records that meet certain criteria by entering YES, followed by valid record selection expressions, followed by the END command.

The following stored procedure creates the CARMDI MDI and contains only the user-supplied portion of the preceding dialogue, entered in uppercase:

```
REBUILD
MDINDEX
NEW
CAR
CARMDI
NO
```

Using a Multi-Dimensional Index in a Query

FOCUS allows you to use an MDI in a TABLE or SQL query. The performance is best when all of the dimensions in the MDI are used in selection criteria for the query.

There are two ways to use an MDI with a TABLE query:

- Lazy OR parameters. For example:

  ```
  IF (or WHERE) field EQ value_1 OR value_2 OR value_3. . .
  ```

- Mask or wildcard characters. For example, the following would show all values that begin with A for the field FIELD:

  ```
  IF (or WHERE) field EQ A*
  ```

  field is a dimension in an MDI.

You can use an MDI with an SQL query by issuing an SQL SELECT statement with a WHERE test using a field of an MDI. For example,

```
SELECT field_1, field_2 FROM table WHERE field_3 = value;
```

where *field_3* is a dimension in an MDI.

Note: AUTOINDEX must be turned on for this feature to be operational.

Querying a Multi-Dimensional Index

FOCUS provides a query command that generates statistics and descriptions for your MDIs. The command ? MDI allows you to display information about MDIs for a given FOCUS/XFOCUS Master File that hosts the target of your MDI.

Syntax How to Query a Multi-Dimensional Index

```
? MDI mastername {mdiname|*} [HOLD [AS holdfile]]
```

where:

mastername

Is the logical name of the Master File. If you do not include any other parameters, a list of all MDI names specified is displayed with the command TARGET_OF in the Access File for this *mastername*. If the Access File for the *mastername* does not have any MDI information, an error message will display.

mdiname

Is the logical name of an MDI. Specifying this parameter displays all the dimensions that are part of this MDI.

mdiname must be specified as TARGET_OF in the Access File for this *mastername*, or an error message will display. If any of the dimensions are involved in a parent-child structure, a tree-like picture will display.

*

Displays a list of all dimensions, by MDI, whose targets are specified inside the Access File for this *mastername*.

HOLD

Saves the output in a text file.

holdfile

Is the file in which the output is saved. If this is not included with the AS phrase, the file is named HOLD.

Using AUTOINDEX to Choose an MDI

When an Access File defines multiple MDIs, retrieval efficiency for a query may become a function of the index it uses to access the data source. The AUTOINDEX facility analyzes each retrieval request and chooses the MDI or B-tree index that provides the best access to the data. You can issue the AUTOINDEX command at the command prompt, in a FOCEXEC, or in a profile. The AUTOINDEX facility can be enabled or disabled. The default is to disable AUTOINDEX, but FOCUS ships with a FOCPARM profile that turns it on when you enter FOCUS.

Analyzing a Retrieval Request With AUTOINDEX

In its analysis, AUTOINDEX considers the following factors:

- The segment most involved in the query.
- The MDI with the most filtering conditions (IF/WHERE selection tests).
- The percent of index dimensions involved in the request from each MDI.
- How close the fields being retrieved are to the target segment.
- The size of each MDI.

If the selection criteria in a request do not involve any MDI fields, FOCUS looks for an appropriate B-tree index to use for retrieval. If a field is both a B-tree index and a dimension in an MDI, the MDI is used for retrieval if two-thirds of the fields in selection tests are dimensions in the MDI. If it is less than two-thirds, the B-tree index is used. If there are multiple B-tree indexes, the one highest in the hierarchy is used.

If everything else is equal, FOCUS uses the first MDI it finds in the Access File.

Syntax How to Enable or Disable AUTOINDEX for Retrieval

```
SET AUTOINDEX {ON|OFF}
```

where:

`ON`

Optimizes MDI retrieval. AUTOINDEX can only be set to ON when the ALL parameter is set to OFF. This value is not the default. However, the FOCPARM file shipped with FOCUS turns AUTOINDEX ON.

`OFF`

Disables the AUTOINDEX facility. No MDI will be used for retrieval. This value is the default.

Note: FOCUS TABLE requests can automatically turn off AUTOINDEX and select the appropriate index for retrieval by indicating the index name in the request itself:

```
TABLE FILE filename.mdiname
```

You can also assure access with a specific MDI by creating an Access File that describes only that index.

Joining to a Multi-Dimensional Index

Joining to an MDI uses the power of the MDI to produce a fast, efficient join. Instead of joining one field in the source file to an indexed field in the target file, you can join to multiple dimensions of the MDI.

When the join is implemented, the answer set from the source file is created, and the values retrieved for the source fields serve as index values for the corresponding dimensions in the target MDI.

You can join to an MDI in two ways:

- Join to all dimensions of a named MDI (MDI join). In the MDI join, you pick the MDI to use in the join.
- Join certain dimensions in a dimensional join. In this type of join, the JOIN engine picks the most efficient MDI based on the dimensions you choose. The dimensional join supports partial joins.

Choosing Source Fields

The source fields must form a one-to-one correspondence with the target dimensions. The MDI engine uses the source field values to get pointers to the target segment of the MDI, expediting data retrieval from the target file.

You can think of the source fields as mirror dimensions. If you put tighter constraints on the mirror dimensions, a smaller answer set is retrieved from the source file, and fewer index I/Os are required to locate the records from the target file. The speed of the join improves dramatically with the speed of retrieval of the source file answer set. Therefore, you can expedite any TABLE request against the source file by incorporating selection criteria on fields that participate in either a B-tree or MDI.

The following formula computes the time for a TABLE request that uses an MDI Join:

```
Total Time = Time to Retrieve the answer set from the source file (Ts)
+ Time to retrieve the MD index pointers (Tp)
+ Time to retrieve data from the target file (Tt)
```

Using a B-tree index or MDI in data retrieval reduces all types of retrieval time, reducing the total retrieval time.

Syntax How to Join to All Dimensions of a Multi-Dimensional Index

```
JOIN field_1 [AND field_2 ...] IN sfile [TAG tag_1]
TO ALL mdiname IN tfile [TAG tag_2][AS joinname]
[END]
```

where:

field_1, field_2

Are the join fields from the source file.

sfile

Is the source Master File.

tag_1, tag_2

Are one- to eight-character names that can serve as unique qualifiers for field names in a request.

mdiname

Is the logical name of the MDI, built on *tfile*, to use in the join.

tfile

Is the target Master File.

joinname

Is a one- to eight-character join name. A unique join name prevents a subsequent join from overwriting the existing join and allows you to selectively clear the join, and serves as a prefix for duplicate field names in a recursive join.

END

Is required to terminate the JOIN command if it is longer than one line.

Syntax How to Create a Dimensional Join

```
JOIN field_1 [AND field_2 ...] IN sfile [TAG tag_1]
TO ALL dim_1 [AND dim_2 ...] IN tfile [TAG tag_2] [AS joinname]
[END]
```

where:

field_1, field_2

Are the join fields from the source file.

sfile

Is the source Master File.

tag_1, tag_2

Are one- to eight-character names that can serve as unique qualifiers for field names in a request.

dim_1, dim_2

Are dimensions in *tfile*.

tfile

Is the target Master File.

joinname

Is a one- to eight-character join name. A unique join name prevents a subsequent join from overwriting the existing join and allows you to selectively clear the join, and serves as a prefix for duplicate field names in a recursive join.

END

Is required to terminate the JOIN command if it is longer than one line.

Reference Guidelines for Choosing a Source Field (Dimensional Joins Only)

- A maximum of four mirror and MDI dimensions can participate in a JOIN command.
- The target of the MDI must be a real segment in the target file.
- The order of the mirror dimensions must match the exact physical order of the MDI (target) dimensions.
- The format of each mirror dimension must be identical to that of the corresponding MDI dimension.
- The ALL attribute is required.

Encoding Values in a Multi-Dimensional Index

FOCUS encodes indexed values any time a field or dimension of an MDI has a MAXVALUES attribute specified or is involved in a parent-child relationship. Encoded values are stored in the MDI file at rebuild time and can be retrieved and decoded with a TABLE request that specifies the MDIENCODING command. The MDIENCODING command allows the user to get output from the MDI file itself without having to read the data source.

Reference Rules for Encoding a Multi-Dimensional Index

The following two rules apply to fields in a TABLE request that uses MDIENCODING:

- Only one MDI can be referred to at a time.

- Only dimensions that are part of the same parent-child hierarchy can be used simultaneously in a request. A dimension that is not part of a parent-child relationship can be used as the field in a request if it has a MAXVALUES attribute.

Syntax How to Retrieve Output From a Multi-Dimensional Index

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

where:

`ON`

Enables retrieval of output from the MDI file without reading the data source.

`OFF`

Requires access of the data source to allow retrieval of MDI values.

Syntax How to Encode a Multi-Dimensional Index

```
TABLE FILE masternamemdiname
request
ON TABLE SET MDIENCODING ON
END
```

where:

masternamemdiname

Is the Master File.

mdiname

Is the logical name of the MDI.

request

Is the TABLE request that decodes the MDI.

Example Encoding a Multi-Dimensional Index

The following examples show correct MDI encoding:

```
TABLE FILE COMPANY.I DATA1
PRINT CITY BY STATE
ON TABLE SET MDIENCODING ON
END
TABLE FILE COMPANY.I DATA1
COUNT CITY
IF STATE EQ NY
ON TABLE SET MDIENCODING ON
END
TABLE FILE COMPANY.I DATA1
PRINT CATEGORY
ON TABLE SET MDIENCODING ON
END
```

The following example is incorrect because CATEGORY is not part of the CITY-STATE hierarchy.

```
TABLE FILE COMPANY.I DATA1
PRINT CITY BY STATE
IF STATE EQ NY
IF CATEGORY EQ RESTAURANT
ON TABLE SET MDIENCODING ON
END
```

Example Using a Multi-Dimensional Index in a Request

The following TABLE request accesses the CAR data source. You know it will use the CARMDI index for retrieval because CARMDI is the only MDI described in the Master File:

```
TABLE FILE CAR
SUM RETAIL_COST DEALER_COST
BY BODYTYPE
-* WHERE Condition utilizing MDI fields:
WHERE (COUNTRY EQ 'JAPAN' OR 'ENGLAND')
AND (CAR EQ 'TOYOTA' OR 'JENSEN' OR 'TRIUMPH')
AND (MODEL EQ 'COROLLA 4 DOOR DIX AUTO'
OR 'INTERCEPTOR III' OR 'TR7')
END
```

Partitioning a Multi-Dimensional Index

If the data source has grown due to the addition of new data partitions, and these partitions need to be added to the MDI, you must perform the following steps:

1. Update the Access File to include the new data partitions.
2. Verify that your MDI is partitioned. Remember that the ADD function of the REBUILD utility cannot be executed on a non-partitioned MDI.
3. Perform the REBUILD, MDINDEX, and ADD on the MDI.

Example Adding a Partition to a Multi-Dimensional Index

The following procedure is a REBUILD dialogue:

1. Issue REBUILD at the command prompt. The REBUILD command invokes the REBUILD utility.

   The following menu displays:

```
Enter option
1. REBUILD (Optimize the database structure)
2. REORG (Alter the database structure)
3. INDEX (Build/modify the database index)
4. EXTERNAL INDEX (Build/modify an external index database)
5. CHECK (Check the database structure)
6. TIMESTAMP (Change the database timestamp)
7. DATE NEW (Convert old date formats to smartdate formats)
8. MDINDEX (Build/modify a multidimensional index.)
9. MIGRATE (Convert FOCUS masters/DBs to FUSION)
```

2. Enter MDINDEX. This command invokes the MDI option of the REBUILD utility.

   The following message appears:

```
NEW/ADD
```

3. Enter ADD. The NEW/ADD option enables you to create a new MDI or to add new data partitions to an existing MDI.

   The following message appears:

```
ENTER THE NAME OF THE MASTER
```

4. Enter the name of the Master File. The Master File contains the segment that is the TARGET_OF the MDI.

   The following message appears:

   ```
   ENTER MD_INDEX LOCATION FILE NAME
   ```

5. Enter the logical name of the MDI.

   The following message appears:

   ```
   ANY RECORD SELECTION TESTS? (YES/NO)
   ```

6. Enter NO to indicate that there are no record selection criteria for the index. The user can limit the index to records that meet certain criteria by entering YES, followed by valid record selection expressions, followed by the END command.

Once the MDI is rebuilt to include the new data partitions, any retrieval query that uses the MDI will use the newly added data partitions within that MDI.

Querying the Progress of a Multi-Dimensional Index

Use the SET MDIPROGRESS command to view messages about the progress of your MDI build. The messages will show the number of data records accumulated for every *n* records inserted into the MDI as it is processed.

Syntax How to Query the Progress of a Multi-Dimensional Index

```
SET MDIPROGRESS = {0|n}
```

where:

n

Is an integer greater than 1000, which displays a progress message for every *n* records accumulated in the MDI build. The default value is 100,000.

0

Disables progress messages.

Displaying a Warning Message

The SET MDICARDWARN command displays a warning message every time a dimension's cardinality exceeds a specified value, offering you the chance to study the MDI build. When the number of equal values of a dimension's data reaches a specified percent, a warning message will be issued. In order for MDICARDWARN to be reliable, the data source should contain at least 100,000 records.

Note: In addition to the warning message, a number displays in brackets. This number is the least number of equal values for the dimension mentioned in the warning message text.

Syntax How to Display a Warning Message

```
SET MDICARDWARN = n
```

where:

n

Is a percentage value from 0 to 50.

NF989: Excel 97

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 97 Reports

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*.

Syntax How to Create a Styled Excel 97 File

To produce an Excel 97 spreadsheet, you 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 Report

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
16			

When using 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 minimize 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.)

NF994: FOCUS SETS From Maintain

The ability to change the setting of specific environmental controls from within a Maintain application gives greater flexibility and control within a Maintain session. It allows some FOCUS environmental variables and system settings to be set on the fly and respected from within the same Maintain session.

Syntax How to Define FOCUS SET Commands Within Maintain

```
SYS_MGR.FOCSET('parm','value');
```

where:

parm

Is the FOCUS command to be set.

value

Is the value of the setting.

Reference FOCUS SET Commands Supported Within Maintain

The following FOCUS SET commands are supported from within Maintain procedures:

```
CDN
COMMITD
DATEDISPLAY
DEFCENT; DFC
LANGUAGE; LANG
MESSAGE
NODATA
TRACEON
TRACEOFF
TRACEUSER
SCROLL
YRTHRESH; YRT
PASS
USER
```

Example Using SET Commands Within Maintain

```
MAINTAIN
COMPUTE MYDATE/YYMD;
SYS_MGR.FOCSET( 'DEFCENT', '21' );
COMPUTE DATE1/YMD='90/01/01';
COMPUTE MYDATE=DATE1;
TYPE "AFTER SETTING DEFCENT='21', MYDATE = <<MYDATE";
END
```

The output is:

```
AFTER SETTING DEFCENT=21, MYDATE = 2190/01/01
```

NF995: Raised Limit for IF-THEN-ELSE Conditions

The limit for the number of nested IF-THEN-ELSE conditions and nested calculations in DEFINE and COMPUTE commands has been raised significantly. The actual number allowed is dependent on the complexity of the expressions being evaluated.

Example Using Nested IF-THEN-ELSE Conditions

The following example has 51 conditions:

```
TABLE FILE CENTORD
PRINT ORDER_NUM AND
COMPUTE ORDERID/I5 = IF ORDER_NUM EQ '28003' THEN 1
ELSE IF ORDER_NUM EQ '28004' THEN 2
ELSE IF ORDER_NUM EQ '28005' THEN 3
ELSE IF ORDER_NUM EQ '28006' THEN 4
ELSE IF ORDER_NUM EQ '28008' THEN 5
ELSE IF ORDER_NUM EQ '28010' THEN 6
ELSE IF ORDER_NUM EQ '28011' THEN 7
ELSE IF ORDER_NUM EQ '28012' THEN 8
ELSE IF ORDER_NUM EQ '28014' THEN 9
ELSE IF ORDER_NUM EQ '28015' THEN 10
ELSE IF ORDER_NUM EQ '28018' THEN 11
ELSE IF ORDER_NUM EQ '28020' THEN 12
ELSE IF ORDER_NUM EQ '28024' THEN 13
ELSE IF ORDER_NUM EQ '28025' THEN 14
ELSE IF ORDER_NUM EQ '28027' THEN 15
ELSE IF ORDER_NUM EQ '28028' THEN 16
ELSE IF ORDER_NUM EQ '28029' THEN 17
ELSE IF ORDER_NUM EQ '28030' THEN 18
ELSE IF ORDER_NUM EQ '28031' THEN 19
ELSE IF ORDER_NUM EQ '28032' THEN 20
```

```
ELSE IF ORDER_NUM EQ '28033' THEN 21
ELSE IF ORDER_NUM EQ '28034' THEN 22
ELSE IF ORDER_NUM EQ '28035' THEN 23
ELSE IF ORDER_NUM EQ '28036' THEN 24
ELSE IF ORDER_NUM EQ '28037' THEN 25
ELSE IF ORDER_NUM EQ '28038' THEN 26
ELSE IF ORDER_NUM EQ '28040' THEN 27
ELSE IF ORDER_NUM EQ '28041' THEN 28
ELSE IF ORDER_NUM EQ '28042' THEN 29
ELSE IF ORDER_NUM EQ '28043' THEN 30
ELSE IF ORDER_NUM EQ '28045' THEN 31
ELSE IF ORDER_NUM EQ '28046' THEN 32
ELSE IF ORDER_NUM EQ '28048' THEN 33
ELSE IF ORDER_NUM EQ '28050' THEN 34
ELSE IF ORDER_NUM EQ '28052' THEN 35
ELSE IF ORDER_NUM EQ '28053' THEN 36
ELSE IF ORDER_NUM EQ '28058' THEN 37
ELSE IF ORDER_NUM EQ '28060' THEN 38
ELSE IF ORDER_NUM EQ '28061' THEN 39
ELSE IF ORDER_NUM EQ '28063' THEN 40
ELSE IF ORDER_NUM EQ '28064' THEN 41
ELSE IF ORDER_NUM EQ '28065' THEN 42
ELSE IF ORDER_NUM EQ '28067' THEN 43
ELSE IF ORDER_NUM EQ '28068' THEN 44
ELSE IF ORDER_NUM EQ '28069' THEN 45
ELSE IF ORDER_NUM EQ '28070' THEN 46
ELSE IF ORDER_NUM EQ '28073' THEN 47
ELSE IF ORDER_NUM EQ '28075' THEN 48
ELSE IF ORDER_NUM EQ '28048' THEN 49
ELSE IF ORDER_NUM EQ '28050' THEN 50
ELSE 99999;
BY ORDER_NUM NOPRINT
WHERE ORDER_NUM GE '28003' AND ORDER_NUM LE '28061'
END
```

The output is:

```
PAGE     1

Order
Number:  ORDERID
-------  -------
28003          1
28004          2
28005          3
28006          4
28008          5
28010          6
28011          7
28012          8
28014          9
28015         10
28018         11
28020         12
28024         13
28025         14
28027         15
28028         16

  .
  .
  .
PAGE     6

Order
Number:  ORDERID
-------  -------
28048         33
28050         34
28052         35
28053         36
28058         37
28060         38
28061         39
```

7.2 New Features

General Enhancements

NF796: Unlimited Nested -INCLUDE Commands
NF802: Long Master File Names
NF824: STRIP Subroutine
NF825: SET PCOMMA
NF826: SQUEEZ Subroutine
NF827: TRIM Subroutine
NF828: SUFFIX = COM/COMT/TABT
NF829: AUTODATE for FOCUS Data Sources
NF880: Exit on Error
NF886: Reusable MSO/CICS Control Blocks
NF914: LMF Site Code Warning Message

Raised Limits

NF767: Long Alphanumeric Fields
NF787: Increased Report Width
NF796: Unlimited Nested -INCLUDE Commands
NF802: Long Master File Names

Reporting Enhancements

NF646: WHERE-Based JOIN
NF788: Tiles
NF789: FORECAST
NF790: ACROSS-TOTAL
NF792: SET CDN Command Enhancement
NF793: Displaying Fractions
NF797: Field-Based Reformatting
NF817: Creating Comma-Delimited Files
NF833: Creating Tab-Delimited Files
NF849: Extended Currency Symbol Support

Data Adapter Enhancements

NF551: AUTOIMS
NF801: DB2 Data Adapter SET SSID Command for MSO
NF898: SQL Support for Date-Time Formats

NF551: AUTOIMS

AUTOIMS is an automated procedure that creates a Master File and FOCPSB for an IMS data source. This procedure operates in three phases:

- In the first phase, AUTOIMS extracts segment and key field information from an IMS DBD and PSB you use to access the data source. This phase creates a basic Master File that contains segment information and descriptions of key fields and search fields.
- In the next phase, AUTOIMS extracts detailed field information from a COBOL FD for each segment. The accuracy of the Master File created by the AUTOIMS procedure depends on having a working, syntactically correct COBOL FD for *every segment listed in the DBD of the IMS data source*.
- In the last phase, AUTOIMS merges the results from the first two phases to create the final Master File and the FOCPSB.

For a detailed description of the Master File and FOCPSB, see your documentation on the IMS/DB Data Adapter. For detailed information about the COBOL FD Translator, see the *COBOL FD Translator for S/390 User's Manual and Installation Guide.*

AUTOIMS Prerequisites

Before you can execute the AUTOIMS procedure, the following prerequisites must be met:

- The COBOL FD Translator, Release 3.0 or higher must be installed and operational.

- IMS Version 6 or higher must be installed and operational, and the DBDLIB and PSBLIB must contain appropriate DBDs and PSBs for the IMS data sources.
- Any supported release of FOCUS for S/390 on MVS or OS/390 must be installed and operational.
- You must have MASTER and FOCPSB data sets to receive the created Master File and FOCPSB members.

Installing AUTOIMS

AUTOIMS is installed during FOCUS installation. No additional installation steps are needed.

Invoking AUTOIMS

AUTOIMS can only be invoked interactively in a FOCUS session. The CLIST that starts the session must allocate the following ddnames:

DDNAME	Allocated to the...
FOCLIB	FOCUS product load library.
USERLIB	COBOL FD Translator load library.
MASTER	PDS that contains Master Files.
FOCEXEC	PDS that contains stored procedures.
ERRORS	PDS that contains Information Builders error messages.

TRF	PDS that contains windows displayed by AUTOIMS that list and let you select members to use in the AUTOIMS procedure.
HOLDMAST	PDS that contains HOLD Master Files.

Example Editing the Sample CLIST for AUTOIMS

Edit the following CLIST to conform to your site's standards before executing it:

```
PROC 0  CONTROL NOLIST NOMSG NOFLUSH NOSYM END(ENDO)
ALLOC F(FOCLIB)   DA('prefix.FOCLIB.LOAD') SHR REUSE
ALLOC F(USERLIB)  DA('prefix.CTF30.LOAD' +
                     'prefix.FUSELIB.LOAD') SHR REUSE
ALLOC F(MASTER)   DA('prefix.MASTER.DATA')  SHR REUSE
ALLOC F(FOCEXEC)  DA('prefix.FOCEXEC.DATA') SHR REUSE
ALLOC F(ERRORS)   DA('prefix.ERRORS.DATA') SHR REUSE
ALLOC F(TRF)      DA('prefix.FOCEXEC.DATA') SHR REUSE
ALLOC F(HOLDMAST) DA('user.HOLDMAST.DATA') SHR REUSE
CALL 'prefix.FOCLIB.LOAD(FOCUS)'
EXIT
```

where:

prefix

Is the high-level qualifier for your site's FOCUS production data sets.

user

Is the high-level qualifier for the HOLDMAST data set. You must have authority to create new members in this data set.

Note: You are also required to have authority to create members in the data sets that will receive the Master Files and FOCPSBs created by AUTOIMS. You enter the names of these data sets on the first window displayed by AUTOIMS.

Procedure How to Start the AUTOIMS Procedure and Log the Parameters

1. From the FOCUS prompt, issue the following command:

   ```
   EX AUTOIMS
   ```

   The AUTOIMS Main Menu displays:

   ```
   Main Menu      Master File Generation Facility for IMS

           PSB Name            =>
           PSBLIB    Dataset   => IMS.V6R1M0.DBT.PSBLIB
           DBDLIB    Dataset   => IMS.V6R1M0.DBT.DBDLIB
           COBOL FD Dataset    => PMSERH.TSO.SRCE

         Descriptions will be members of:
           FOCPSB Target PDS => PMSSAE.FOCPSB.DATA
           Master Target PDS => PMSSAE.MASTER.DATA
           Preliminary   PDS => PMSSAE.PRE.DATA
           Translator    PDS => PMSSAE.TRANS.DATA




   Parm File => PMSSAE.FOCPSB.DATA

   PF   1=Help    3=Exit    4=Log    5=Members    6=TED
       9=Header 10=Messages  11=Debugging 12=Audit
   ```

2. In this window, supply the name of the PSB and data sets AUTOIMS requires.

Once the correct information is entered, you can log it to a file so that it will display automatically in subsequent executions of AUTOIMS.

3. Press Enter to proceed. AUTOIMS displays a message to indicate that it is processing your menu selections.

Reference Required Data Set Names for AUTOIMS

AUTOIMS requires the following data sets:

- The PSBLIB. By default, AUTOIMS supplies the name *userid*.PSBLIB.DATA, where *userid* is your user ID.
- The DBDLIB. By default, AUTOIMS supplies the name *userid*.DBDLIB.DATA, where *userid* is your user ID.
- The data set that contains the COBOL FDs. Each COBOL FD that will be used in generating a Master File must be stored as a member in this PDS. Each COBOL FD can have only one 01 level entry.
- The data set that will contain the FOCPSBs created by AUTOIMS. By default, AUTOIMS supplies the name *userid*.FOCPSB.DATA, where *userid* is your user ID. You must have authority to create new members in this data set.
- The data set that will contain the Master Files created by AUTOIMS. By default, AUTOIMS supplies the name *userid*.MASTER.DATA, where *userid* is your user ID. You must have authority to create new members in this data set.
- The preliminary Master File library. This transitional work library receives the output of phase one of AUTOIMS processing and is useful for troubleshooting.

- The Translator library. This transitional work library receives the output of phase two of AUTOIMS processing and is useful for troubleshooting.

If the default data set names are not the ones you need, type the correct names over them.

ProcedureHow to Select the PSB Name

If you know the PSB name you want AUTOIMS to use, you can enter it directly in the window.

You can also select the PSB name from the PSBLIB:

1. With the cursor on the PSB name line of the AUTOIMS Main Menu, press PF5. The Member List window displays:

```
Main Menu     Master File Generation Facility for IMS

        PSB Name           =>
        PSBLIB   Dataset   => IMS.V6R1M0.DBT.PSBLIB
        DBDLIB   Dataset   => IMS.V6R1M0.DBT.DBDLIB
        COBOL FD Dataset   => PMSERH.TSO.SRCE

      Descriptions will be members of:
        FOCPSB Target PDS => PMSSAE.FOCPSB.DATA
        MASTER Target PDS => PMSSAE.MASTER.DATA
        PRELIMINARY   PDS => PMSSAE.PRE.DATA
        TRANSLATOR    PDS => PMSSAE.TRANS.DATA

Member DSN=> IMS.V6R1M0.DBT.PSBLIB

Parm File => PMSSAE.FOCPSB.DATA

PF  1=Help   3=Exit   4=Log
```

```
Member List
AIHDPSB
AIRPSB
AIR2PSB
ALLPSB
BELLPSB1
BIGPSB31
BIG31L
BMHIPSB
B1001047
CARIMSLD
CARIMSRD
CARIMSWR
CARLOAD
CARPSB
CARREAD
CARREED
(MORE)
```

2. Page down until the cursor is under the PSB you need, and press Enter. The PSB is entered on the PSB name line:

```
Main Menu      Master File Generation Facility for IMS

         PSB Name            => DI21PSB
         PSBLIB    Dataset   => IMS.V6R1M0.DBT.PSBLIB
         DBDLIB    Dataset   => IMS.V6R1M0.DBT.DBDLIB
         COBOL FD Dataset    => PMSERH.TSO.SRCE

       Descriptions will be members of:
         FOCPSB Target PDS => PMSSAE.FOCPSB.DATA
         Master Target PDS => PMSSAE.MASTER.DATA
         Preliminary   PDS => PMSSAE.PRE.DATA
         Translator    PDS => PMSSAE.TRANS.DATA




Parm File => PMSSAE.FOCPSB.DATA

PF  1=Help   3=Exit   4=Log   5=Members   6=TED
    9=Header 10=Messages  11=Debugging 12=Audit
```

You can view the member list for any of the data sets listed on the Main Menu by placing the cursor under the data set name and pressing PF5. You can then:

- Select a member and open it for editing in TED by pressing PF6.
- Close the Member List window without making a selection by pressing PF3.

ProcedureHow to Save Default AUTOIMS Parameters

AUTOIMS can save the data set names and the PSB name you enter on the Main Menu. In future executions these values will display as the default values on the AUTOIMS Main Menu.

- When the information you want to log is displayed on the main window, press PF4. The parameters are saved in the Parameter Log file, which is member IMS$PRM in the FOCPSB data set. The first time you press PF4, AUTOIMS creates the log file. If you press PF4 again, AUTOIMS updates the file with the values currently displayed:

```
Main Menu     Master File Generation Facility for IMS

        PSB Name            => DI21PSB
        PSBLIB   Dataset    => IMS.V6R1M0.DBT.PSBLIB
        DBDLIB   Dataset    => IMS.V6R1M0.DBT.DBDLIB
        COBOL FD Dataset    => PMSERH.TSO.SRCE

      Descriptions will be members of:
        FOCPSB Target PDS => PMSSAE.FOCPSB.DATA
        Master Target PDS => PMSSAE.MASTER.DATA
        Preliminary   PDS => PMSSAE.PRE.DATA
        Translator    PDS => PMSSAE.TRANS.DATA

Parm File => PMSSAE.FOCPSB.DATA
*** PARMS LOGGED ***
PF  1=Help   3=Exit   4=Log   5=Members   6=TED
    9=Header 10=Messages  11=Debugging 12=Audit
```

AUTOIMS Master File Generation Window

When all of the necessary information is displayed on the Main Menu window, press Enter. The Master File Generation window displays:

```
AUTOIMS - ENTER MFD NAME TO GENERATE A NEW MASTER
PSB   : DI21PSB                                GEN DATE:  12/09/98  08.36610
PSBLIB: IMS.V6R1M0.DBT.PSBLIB
DBDLIB: IMS.V6R1M0.DBT.DBDLIB

 MFD NAME  TYPE PCB# DBD NAME SEG NO SEG NAME FOCPSB   COBOL FD  ERROR/WARNING
 --------  ---- ---- -------- ------ -------- -------- --------  -------------
           DB   0001 DI21PART 0001   PARTROOT DI211229
                              0002   STANINFO DI211229
                              0003   STOKSTAT DI211229
                              0004   CYCCOUNT DI211229
                              0005   BACKORDR DI211229









PF  1=Help      3=Exit      5=Ignore Warnings      PF7=Up  PF8=Down
```

Note that the FOCPSB name is the name from the last time you logged parameters. The FOCPSB created at the end of AUTOIMS processing will have the correct name for the IMS PSB referenced.

Procedure How to Enter Information on the AUTOIMS Master File Generation Window

1. Enter a name for the new Master File in the MFD NAME column.

2. Enter the name of the COBOL FD for each segment listed. On the following screen, the Master File name is DI210307 and the COBOL FD names are the same as the segment names. Note that it is a requirement of the COBOL FD Translator that each segment in the DBD have a separate COBOL FD:

```
AUTOIMS - ENTER MFD NAME TO GENERATE A NEW MASTER
PSB    : DI21PSB                                  GEN DATE:  12/09/98  08.36610
PSBLIB: IMS.V6R1M0.DBT.PSBLIB
DBDLIB: IMS.V6R1M0.DBT.DBDLIB

 MFD NAME  TYPE PCB# DBD NAME SEG NO SEG NAME FOCPSB   COBOL FD  ERROR/WARNING
 --------  ---- ---- -------- ------ -------- -------- --------  -------------
 DI210307  DB   0001 DI21PART 0001   PARTROOT DI211229 PARTROOT
                             0002   STANINFO DI211229 STANINFO
                             0003   STOKSTAT DI211229 STOKSTAT
                             0004   CYCCOUNT DI211229 CYCCOUNT
                             0005   BACKORDR DI211229 BACKORDR

PF  1=Help      3=Exit      5=Ignore Warnings      PF7=Up  PF8=Down
```

3. Press Enter.

If AUTOIMS detects a problem, such as a Master File name that already exists, it places a message in the ERROR/WARNING column. In the following example, the Master File name already exists:

```
AUTOIMS - ENTER MFD NAME TO GENERATE A NEW MASTER
PSB    : DI21PSB                                  GEN DATE:  12/09/98  08.36610
PSBLIB: IMS.V6R1M0.DBT.PSBLIB
DBDLIB: IMS.V6R1M0.DBT.DBDLIB

 MFD NAME  TYPE PCB# DBD NAME SEG NO SEG NAME FOCPSB   COBOL FD  ERROR/WARNING
 --------  ---- ---- -------- ------ -------- -------- --------  -------------
 DI210307  DB   0001 DI21PART 0001   PARTROOT DI210307 PARTROOT  MFD EXISTS
                             0002   STANINFO DI210307 STANINFO
                             0003   STOKSTAT DI210307 STOKSTAT
                             0004   CYCCOUNT DI210307 CYCCOUNT
                             0005   BACKORDR DI210307 BACKORDR

PF  1=Help      3=Exit      5=Ignore Warnings      PF7=Up  PF8=Down
 ERRORS FOUND:   0 .  WARNINGS ISSUED:   1 .
```

- To ignore the warning and overwrite the existing Master File, Press PF5.
- To supply a new name, type over the Master File name and press Enter.

If the PSB has more than one PCB, you must enter a Master File name for each PCB and supply COBOL FD names for each segment in all of the PCBs. If the PSB contains non-database PCBs, they will be handled automatically by AUTOIMS.

The following is an example of a PSB that contains two database PCBs:

```
AUTOIMS - ENTER MFD NAME TO GENERATE A NEW MASTER
PSB    : TEMPSB                                    GEN DATE:  03/07/01  08.33610
PSBLIB: PMS.DBT.V6R1M0.PSBLIB
DBDLIB: PMS.DBT.V6R1M0.DBDLIB

 MFD NAME  TYPE PCB# DBD NAME SEG NO SEG NAME FOCPSB   COBOL FD  ERROR/WARNING
 --------  ---- ---- -------- ------ -------- -------- --------  -------------
 TEMP1     DB   0001 CARDBD   0001   ORIGIN            ORIGIN
                              0002   COMP              COMP
                              0003   CARREC            CARREC
                              0004   BODY              BODY
                              0005   SPECS             SPECS
                              0006   WARANT            WARANT
                              0007   EQUIP             EQUIP
 TEMP2     DB   0002 TACCOUNT 0001   ACMASTER          ACMASTER
                              0005   ACBLKFED          ACBLKFED
                              0014   ACMEMO            ACMEMO



PF  1=Help      3=Exit      5=Ignore Warnings      PF7=Up  PF8=Down
```

Two Master File names are entered, and a COBOL FD name is supplied for each segment in both PCBs. Note that AUTOIMS processing time increases as the number of PCBs increases.

COBOL FD Translator Options Window

Next AUTOIMS displays the COBOL FD Translator Options window:

```
                    COBOL FD Translator - Version 3.0.2
  COBOL FD Dataset Name => PMSERH.TSO.SRCE

       MFD Dataset Name => PMSSAE.MASTER.DATA

 Field Name Length              => 30      (12,30)
 Skip 'n' Hyphens in COBOL Name => 0       (0,1,2,3)
 Remove Hyphens or Use Underbars=> U       (Remove,Underbars)
 Generate GROUP Fields          => A       (All,None,Printable only)
 Generate REDEFINE Fields       => C       (Comments,None,Segments)
 Generate LEVEL 88 as Comments  => N       (Yes,No)
 Describe Occurs as Segments    => Y       (Yes,No)
 Zoned Numeric Field Usage      => A       (Packed,Alpha)
 Numeric Field Edit Options     =>         (S,C,B,R,M,N,L combinations)

 1=Help       2=Read Me    3=Top
```

In many cases the default options are sufficient. However, some situations require changes to these defaults. For a detailed discussion of the options, refer to the *COBOL FD Translator for S/390 User's Manual and Installation Guide.*

When you press Enter, AUTOIMS displays a message indicating that it is processing your request.

Completing AUTOIMS Processing

The Main Menu displays with a completion message when AUTOIMS finishes processing:

```
Main Menu     Master File Generation Facility for IMS

        PSB Name           => DI21PSB
        PSBLIB   Dataset   => IMS.V6R1M0.DBT.PSBLIB
        DBDLIB   Dataset   => IMS.V6R1M0.DBT.DBDLIB
        COBOL FD Dataset   => PMSERH.TSO.SRCE

      Descriptions will be members of:
        FOCPSB Target PDS => PMSSAE.FOCPSB.DATA
        Master Target PDS => PMSSAE.MASTER.DATA
        Preliminary   PDS => PMSSAE.PRE.DATA
        Translator    PDS => PMSSAE.TRANS.DATA




Parm File => PMSSAE.FOCPSB.DATA
Processing complete
PF  1=Help   3=Exit   4=Log   5=Members   6=TED
    9=Header 10=Messages  11=Debugging 12=Audit
```

Reference AUTOIMS Main Menu PF Keys

Several PF key actions are available from the Main Menu:

PF Key	Action
PF1	Displays a help file.
PF3	Exits from the active window.
PF4	Updates the parameter log file with the names entered on the Main Menu.
PF5	Opens a Member List window for the selected data set.
PF6	Opens a member from the Member List window for editing or viewing in TED.
PF9	Opens the file that contains the parameter list passed by AUTOIMS to the COBOL FD Translator.
PF12	Displays a report about the Master File creation process, including detailed descriptions of the fields inserted.

Note that PF10 and PF11 are used by the COBOL FD Translator and do not supply useful information regarding AUTOIMS.

Accessing the Generated Master File and FOCPSB

You can list data set members and open them in TED directly from the AUTOIMS Main Menu.

ProcedureHow to Access the Generated Master File and FOCPSB

1. Place the cursor under the name of the data set whose members you want to list and press PF5. The Member List window displays. It lists the members of the Master Target data set:

```
Main Menu     Master File Generation Facility for IMS      Member List
                                                           (MORE)
        PSB Name          => DI21PSB                       DATETIM2
        PSBLIB   Dataset  => IMS.V6R1M0.DBT.PSBLIB         DB2CAT
        DBDLIB   Dataset  => IMS.V6R1M0.DBT.DBDLIB         DB2SYSTB
        COBOL FD Dataset  => PMSERH.TSO.SRCE               DEDUCT
                                                           DEPTEMP
      Descriptions will be members of:                     DI21MFD
        FOCPSB Target PDS => PMSSAE.FOCPSB.DATA            DI21PART
        MASTER Target PDS => PMSSAE.MASTER.DATA            DI210307
        PRELIMINARY   PDS => PMSSAE.PRE.DATA               DI211220
        TRANSLATOR    PDS => PMSSAE.TRANS.DATA             DI211227
                                                           DI211229
                                                           DTTRANS
                                                           ECOURSE
Member DSN=> PMSSAE.MASTER.DATA                            EDUCFILE
                                                           EEDUC
Parm File => PMSSAE.FOCPSB.DATA                            EEMP1
                                                           (MORE)
PF  1=Help   3=Exit   4=Log              6=TED   9=TED New
```

2. To open a member in TED, place the cursor under the member name and press PF6.

   The following example shows the Master File just created by AUTOIMS, member DI210307 in the Master Target PDS:

```
$$$ CREATED BY AUTOIMS ON 03/07/01 AT 15.29.22 BY PMSSAE
$$$ FROM PSB : DI21PSB  DBD: DI21PART GEN DATE: 12/09/98 08.36610
$$$ IN PSBLIB: IMS.V6R1M0.DBT.PSBLIB
$$$ IN DBDLIB: IMS.V6R1M0.DBT.DBDLIB
FILENAME=DI211229,SUFFIX=IMS,$
 SEGNAME=PARTROOT,SEGTYPE=S2,PARENT=        ,$
  GROUP=PARTROOT_IO                      ,ALIAS=            ,A50
,A050,$
  FIELD=PARTKEY                          ,ALIAS=PARTKEY.KEY ,A17
,A017,$
  FIELD=FILLER                           ,ALIAS=            ,A33
,A033,$
 SEGNAME=STANINFO,SEGTYPE=S2,PARENT=PARTROOT,$
  GROUP=STANINFO_IO                      ,ALIAS=            ,A85
,A085,$
  FIELD=STANKEY                          ,ALIAS=STANKEY.KEY ,A2
,A002,$
  FIELD=FILLER                           ,ALIAS=            ,A83
,A083,$
 SEGNAME=STOKSTAT,SEGTYPE=S2,PARENT=PARTROOT,$
```

```
  GROUP=STOKSTAT_IO                      ,ALIAS=            ,A140
,A140,$
  FIELD=STOCKEY                          ,ALIAS=STOCKEY.KEY ,A16
,A016,$
  FIELD=FILLER                           ,ALIAS=            ,A124
,A124,$
 SEGNAME=CYCCOUNT,SEGTYPE=S2,PARENT=STOKSTAT,$
  GROUP=CYCCOUNT_IO                      ,ALIAS=            ,A25
,A025,$
  FIELD=CYCCKEY                          ,ALIAS=CYCLKEY.KEY ,A2
,A002,$
  FIELD=FILLER                           ,ALIAS=            ,A23
,A023,$
 SEGNAME=BACKORDR,SEGTYPE=S2,PARENT=STOKSTAT,$
  GROUP=BACKORDR_IO                      ,ALIAS=            ,A75
,A075,$
  FIELD=BACKKEY                          ,ALIAS=BACKKEY.KEY ,A10
,A010,$
  FIELD=FILLER                           ,ALIAS=            ,A65
,A065,$
```

The following example shows the FOCPSB just created by AUTOIMS, member DI21PSB in the FOCPSB Target PDS. Note that the name of the FOCPSB is the same as the name of the PSB, and the PCBNAME attribute supplies the name of the Master File:

```
FOCPSB=EXTENDED,$             PSBNAME=DI21PSB
PCBNAME=DI210307,PCBTYPE=DB  ,$ 0001 DI21PART
```

Warning: Comments (lines starting with the dollar sign) are not supported in the FOCPSB.

You can edit the file if necessary:

- To close it without editing, press PF3.

- To close it and save any edits you made, type FILE on the command line and press Enter.

To close the Member List window, press PF3.

Procedure How to Create a New Member from the AUTOIMS Main Menu

1. With the member list for one of the main menu data sets open, press PF9. The following window displays:

```
Main Menu     Master File Generation Facility for IMS

        PSB Name           => DI21PSB
        PSBLIB   Dataset   => IMS.V6R1M0.DBT.PSBLIB
        DBDLIB   Dataset   => IMS.V6R1M0.DBT.DBDLIB
        COBOL FD Dataset   => PMSERH.TSO.SRCE

      Descriptions will be members of:
        FOCPSB Target PDS => PMSSAE.FOCPSB.DATA
        Master Target PDS => PMSSAE.MASTER.DATA
        Preliminary   PDS => PMSSAE.PRE.DATA
        Translator    PDS => PMSSAE.TRANS.DATA


      Enter Member Name   =>


Parm File => PMSSAE.FOCPSB.DATA

PF  1=Help   3=Return 4=      5=          6=
```

2. Enter a name for the new member and press Enter.

Generated Master File

In the generated Master File:

- The FILENAME value is the name of the Master File.
- The SUFFIX is IMS.
- The segment names are the segment names from the IMS PSB.
- The group names are the level 01 names from the COBOL FD.
- Key field and search field names are taken from the IMS PSB; these are the only fields that have ALIAS values in the generated Master File. Names for other fields are taken from the COBOL FD.
- Format conversions are described in the *COBOL FD Translator for S/390 User's Manual and Installation Guide.*

Reference Using the Generated Master File

You may need to manually edit the generated Master File in the following situations:

- There are more than 64 segments (because of REDEFINES and OCCURS segments).
- There are multiple record types. You need to identify the RECTYPE field and the associated values.
- There are OCCURS segments and you want to add the ORDER field to the description.
- You want to add ALIAS values to fields that are not key or search fields.
- There are embedded REDEFINES declarations.
- To edit date formats and format options.

NF646: WHERE-Based JOIN

The WHERE-based (conditional) join facility gives you the flexibility to define a join based on any condition, not just equality between field values. In addition, the host and cross-referenced join fields do not require matching formats and the cross-referenced field does not need to be indexed.

Note: If your join is based on equality, it is more efficient to use the traditional equi-join.

You can specify a conditional JOIN at the command level, in a procedure, or in a Master File. If you define the join in a Master File, only the specific segment named in the Master File participates in the join, but if you issue a dynamic JOIN command, the entire cross-referenced data source participates in the join.

The conditional JOIN command supports FOCUS data sources and relational, VSAM, fixed-format sequential, relational, Adabas, and IMS data sources. Because each data source differs in its ability to handle complex WHERE criteria, the optimization of the WHERE-based JOIN syntax differs depending on the specific data sources involved in the join and the complexity of the WHERE criteria.

Conditional joins in the Master File are supported for relational data sources only. The conditions are considered virtual fields in the Master File.

Syntax How to Issue a Conditional JOIN Command

```
JOIN FILE from_file AT from_field [TAG from_tag] [WITH fieldname]
     TO [ALL|ONE]
     FILE to_file AT to_field [TAG to_tag]
     [AS as_name]
     [WHERE expression1 ;
     WHERE expression2 ;
     ...               ; ]
END
```

where:

from_file

Is the host Master File.

from_field

Is the field name in the host Master File whose segment will be joined to the cross-referenced data source. It can be any field in the segment. It must reside in the lowest level segment that will be referenced.

from_tag

Is the optional tag name that is used as a unique qualifier for fields and aliases in the host data source.

fieldname

Is a real field name used to assign a segment location for a virtual field. Required when issuing a DEFINE field-based WHERE-based JOIN.

ALL

Describes a one-to-many relationship between the *from_file* and *to_file*.

`ONE`

Describes a one-to-one relationship between the *from_file* and *to_file*. **Note:** If you specify a unique join when the relationship between the host and cross-referenced files is one-to-many, the results will be unpredictable.

`to_file`

Is the cross-referenced Master File.

`to_field`

Is the join field name in the cross-referenced Master File. It can be any field in the segment.

`to_tag`

Is the optional tag name that is used as a unique qualifier for fields and aliases in the cross-referenced data source.

`as_name`

Is the name associated with the JOIN.

`expression1, expression2`

Are any expressions valid in a DEFINE FILE command. All of the fields used in all of the expressions the expressions must lie on a single path.

Note: Single line JOIN syntax is not supported. The END command is required.

Example Using the WHERE-based JOIN Command

The following example joins the MOVIES and VIDEOTRK 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 FEE     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:

```
                          Last         Release   Years
Title                     Transaction  Date      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
```

Syntax How to Define a Conditional Join in the Master File

```
SEGNAME=seg,  SEGTYPE=styp,  PARENT=parseg,
        CRFILE=xmfd,  [CRSEG=xseg,  ]
        JOIN_WHERE=expression; [JOIN_WHERE=expression; ...] ,$
```

where:

seg

Is the segment name for the joined segment.

styp

Is the segment type for the joined segment. Can be DKU, DKM, KU, or KM as with traditional cross-references in the Master File. **Note:** If you specify a unique join when the relationship between the host and cross-referenced files is one-to-many, the results will be unpredictable.

parseg

Is the parent segment name.

xmfd

Is the cross-referenced Master File.

xseg

Is the cross-referenced segment, if *seg* is not the same name as the SEGNAME in the cross-referenced Master File.

expression

Is any expression valid in a DEFINE FILE command. All of the fields referenced in all of the expressions must lie on a single path.

Example Conditionally Joining Two DB2 Data Sources in a Master File

The following Master File for the EMPINFO table contains a conditional join to the PAYINFO table. The conditions specify that the job code starts with the letter A and the employee IDs are equal in the two tables:

```
FILENAME=EMPINFO ,SUFFIX=SQLDS,$

SEGNAME=EMPINFO  ,SEGTYPE=S0,$
 FIELD=EMP_ID         ,ALIAS=EID          ,USAGE=A9      ,ACTUAL=A9      ,$
 FIELD=LAST_NAME      ,ALIAS=LN           ,USAGE=A15     ,ACTUAL=A15     ,$
 FIELD=FIRST_NAME     ,ALIAS=FN           ,USAGE=A10     ,ACTUAL=A10     ,$
 FIELD=HIRE_DATE      ,ALIAS=HDT          ,USAGE=YMD     ,ACTUAL=DATE    ,$
 FIELD=DEPARTMENT     ,ALIAS=DPT          ,USAGE=A10     ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL          ,USAGE=P9.2    ,ACTUAL=P4      ,$
 FIELD=CURR_JOBCODE   ,ALIAS=CJC          ,USAGE=A3      ,ACTUAL=A3      ,$
 FIELD=ED_HRS         ,ALIAS=OJT          ,USAGE=F6.2    ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN     ,ALIAS=BONUS_PLAN   ,USAGE=I4      ,ACTUAL=I4      ,$
FIELD=HIRE_DATE_TIME,ALIAS=HDTT           ,USAGE=HYYMDm  ,ACTUAL=HYYMDm  ,$
FIELD=HIRE_TIME      ,ALIAS=HT            ,USAGE=HHIS    ,ACTUAL=HHIS    ,$

SEGNAME=PAYINFO, SEGTYPE=KM, PARENT = EMPINFO,
CRFILE = PAYINFO,
JOIN_WHERE = JOBCODE LIKE 'A%';
JOIN_WHERE = EMP_ID EQ PAYEID; ,$
```

The EMPINFO Access File follows:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1, WRITE = YES,
     DBSPACE = PMSDB.PMSTSP1 ,$
```

The PAYINFO Access File follows:

```
SEGNAME = PAYINFO, TABLENAME = "USER1"."PAYINFO", KEYS = 2,
    WRITE=YES, DBSPACE = PMSDB.PMSTSP1,$
```

The following request prints fields from both tables:

```
TABLE FILE EMPINFO
PRINT SALARY JOBCODE
BY LAST_NAME BY FIRST_NAME
END
```

The output is:

```
LAST_NAME        FIRST_NAME           SALARY  JOBCODE
---------        ----------           ------  -------
BANNING          JOHN             $29,700.00  A17
CROSS            BARBARA          $27,062.00  A17
                                  $25,775.00  A16
GREENSPAN        MARY              $9,000.00  A07
IRVING           JOAN             $24,420.00  A14
                                  $26,862.00  A15
SMITH            RICHARD           $9,500.00  A01
STEVENS          ALFRED           $10,000.00  A07
                                  $11,000.00  A07
```

Conditional JOIN Processing

A WHERE in a JOIN is in effect only when a TABLE request references a cross-referenced segment or its children. If the request makes no such reference, 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 names used in the AT phrases are not used to cause the link. They are simply used as segment references.

Note: If there is no WHERE in effect, the join will produce a Cartesian product.

The conditional JOIN command can reference any field in the joined data source and any field in the parent segment or higher on the parent's path.

When active, these JOIN expressions screen the segment they reside on (the child or joined segment). That is, if no child segment passes the expression, the JOIN follows the rules of the SET ALL command when no child segment exists. Unlike TABLE WHERE clauses, JOIN_WHERE screening does not automatically screen the parent segment when SET ALL=ON.

To issue a DEFINE-based conditional join, the KEEPDEFINES setting, described in *Preserving Virtual Fields Using the KEEPDEFINES Parameter*, must be ON. You then must create all virtual fields before issuing the join. This differs from traditional DEFINE-based joins in which the virtual field is created after the JOIN command is issued. In addition, a virtual field can be part of the JOIN syntax or WHERE criteria.

Preserving Virtual Fields During Join Parsing

By default, a JOIN command clears all DEFINE FILE commands for the host data source and the joined structure. Two methods are available for preserving virtual fields during join parsing.

Preserving Virtual Fields Using the KEEPDEFINES Parameter

Setting KEEPDEFINES to ON reinstates virtual field definitions after the parsing of a JOIN command. With this setting, every time you issue a JOIN command, virtual field definitions that were active prior to the JOIN are saved and reactivated after the join is performed. The set of virtual fields active prior to each join is called a *context*. Each new JOIN command creates a new context for virtual field definitions.

Each new context creates a new layer or command environment. When you first enter the new environment, all of the virtual fields defined in the previous layer are available in the new layer. Overwriting or clearing a virtual field definition affects only the current layer. When you return to the previous layer, its virtual field definitions are intact.

If you redefine an active virtual field after a JOIN, its new definition replaces the prior definition within the context of the JOIN command. When the join is cleared, the old definition is re-activated. New DEFINE commands issued after the JOIN constitute a new context. In each context, all virtual fields of all prior contexts are accessible. In this way, JOIN and DEFINE commands create a stack of contexts.

Clearing a join by issuing the JOIN CLEAR *join_name* command removes all of the contexts that were created after the JOIN *join_name* command was issued.

Syntax How to Preserve Virtual Fields With KEEPDEFINES

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

where:

OFF

Does not preserve virtual fields issued prior to a join. This is the default value

ON

Preserves virtual fields during join parsing.

Example Preserving Virtual Fields During Join Parsing With KEEPDEFINES

The first virtual field, DAYSKEPT, is defined prior to issuing any joins, but after setting KEEPDEFINES to ON. DAYSKEPT is the number of days between the return date and rental date for videotape:

```
SET KEEPDEFINES = ON
DEFINE FILE VIDEOTRK
DAYSKEPT/I5 = RETURNDATE - TRANSDATE;
END
```

The ? DEFINE query command shows that this is the only virtual field defined at this point:

```
FILE     FIELD NAME                      FORMAT  SEGMENT   VIEW        TYPE
VIDEOTRK DAYSKEPT                        I5            4
```

The following request prints all transactions in which the number of days kept is two:

```
TABLE FILE VIDEOTRK
PRINT MOVIECODE TRANSDATE RETURNDATE DAYSKEPT
WHERE DAYSKEPT EQ 2
END
```

The first few lines of output show that each return date is two days after the transaction date:

```
MOVIECODE  TRANSDATE  RETURNDATE  DAYSKEPT  ACTUAL_DAYS
---------  ---------  ----------  --------  -----------
001MCA     91/06/27   91/06/29           2            2
692PAR     91/06/27   91/06/29           2            2
259MGM     91/06/19   91/06/21           2            2
```

Now, the VIDEOTRK data source is joined to the MOVIES data source. The ? DEFINE query shows that the join did not clear the DAYSKEPT virtual field:

```
JOIN  MOVIECODE IN VIDEOTRK TO ALL MOVIECODE IN MOVIES AS J1
? DEFINE
FILE     FIELD NAME                      FORMAT  SEGMENT   VIEW        TYPE
VIDEOTRK DAYSKEPT                         I5           4
```

Next a new virtual field, YEARS, is defined for the join between VIDEOTRK and MOVIES:

```
DEFINE FILE VIDEOTRK
YEARS/I5 = (TRANSDATE - RELDATE)/365;
END
```

The ? DEFINE query shows that the virtual field created prior to the join was not cleared by this new virtual field because it was in a separate context:

```
FILE     FIELD NAME                      FORMAT  SEGMENT   VIEW        TYPE
VIDEOTRK DAYSKEPT                         I5           4
VIDEOTRK YEARS                            I5           5
```

Next, the field DAYSKEPT is re-defined so that it is the actual number of days plus one:

```
DEFINE FILE VIDEOTRK
DAYSKEPT/I5 = RETURNDATE - TRANSDATE + 1;
END
```

The ? DEFINE query shows that there are two versions of the DAYSKEPT virtual field. However, YEARS was cleared because it was in the same context (after the join) as the new version of DAYSKEPT and the DEFINE command did not specify the ADD option:

```
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

Occasionally, new code needs to be added to an existing application. When adding code, there is always the possibility of over-writing existing virtual fields by reusing their names inadvertently.

The DEFINE FILE SAVE command forms a new context for virtual fields, which can then be removed with DEFINE FILE RETURN. Each new context creates a new layer or command environment. When you first enter the new environment, all of the virtual fields defined in the previous layer are available in the new layer. Overwriting or clearing a virtual field definition affects only the current layer. When you return to the previous layer using DEFINE FILE RETURN, its virtual field definitions are intact.

Therefore, all the virtual fields that are created in the new application can be removed before returning to the calling application, without affecting existing virtual fields in that application.

Syntax How to Use DEFINE FILE SAVE and DEFINE FILE RETURN

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

where:

`SAVE`

Creates a new context for virtual fields.

`mfd`

Is the name of the Master File for which a new context is created.

RETURN

Clears the current context if it was created by DEFINE FILE SAVE and restores the previous context.

Note: A JOIN can be issued after a DEFINE FILE SAVE. In order to clear the join context, you must issue a JOIN CLEAR command. If only DEFINE FILE and DEFINE FILE ADD commands were issued after a DEFINE FILE SAVE, you can clear them by issuing a DEFINE FILE RETURN.

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
? 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.

Clearing Joins

You can clear joins by issuing the JOIN CLEAR command. The effect of the JOIN CLEAR command depends on whether any conditional joins exist:

- If no conditional joins exist, the JOIN CLEAR command clears all virtual fields defined for the host data source and the joined structure.
- If conditional joins exist but were issued prior to the join you want to clear, the JOIN CLEAR command clears only the specified join. Any virtual fields saved in the context of a join that is cleared will also be cleared.
- If conditional joins exist and were issued subsequent to the join you want to clear, or if the join you want to clear is a conditional join, the JOIN CLEAR command clears the specified join and all subsequent joins issued for the same host file.

The JOIN CLEAR * command clears every join that was issued, along with its associated virtual fields. However, virtual fields defined in the null context (prior to any joins) remain in effect.

Example Clearing Joins

The following request creates three joins using VIDEOTRK as the host data source. The first two are conditional (JW1, JW2), and the third join is unconditional (J1):

```
JOIN FILE VIDEOTRK AT PRODCODE TO ALL
     FILE GGSALES  AT PCD AS JW1
WHERE PRODCODE NE PCD;
END
JOIN  FILE VIDEOTRK AT TRANSDATE TO ALL
      FILE MOVIES   AT RELDATE   AS JW1
WHERE (TRANSDATE - RELDATE)/365 GT 10;
END
JOIN MOVIECODE IN VIDEOTRK TO MOVIECODE IN MOVIES AS J1
```

The next request creates a conditional join (JW3) using MOVIES as the host data source:

```
JOIN  FILE MOVIES   AT MOVIECODE TO ONE
      FILE VIDEOTRK AT TRANSDATE AS JW2
WHERE (TRANSDATE - RELDATE)/365 LT 2;
END
```

The last request creates a third conditional join (JW4) that uses VIDEOTRK as the host data source:

```
JOIN  FILE VIDEOTRK AT LASTNAME  TO ALL
      FILE EMPLOYEE AT LAST_NAME AS JW3
WHERE LASTNAME GE LAST_NAME;
END
```

Following is the output of the ? JOIN query after executing these joins:

```
? JOIN
 JOINS CURRENTLY ACTIVE

HOST                            CROSSREFERENCE
FIELD        FILE     TAG       FIELD        FILE     TAG      AS       ALL WH
-----        ----     ---       -----        ----     ---      --       --- --
PRODCODE     VIDEOTRK           PCD          GGSALES           JW1       Y   Y
TRANSDATE    VIDEOTRK           RELDATE      MOVIES            JW2       Y   Y
MOVIECODE    VIDEOTRK           MOVIECODE    MOVIES            J1        N   N
MOVIECODE    MOVIES             TRANSDATE    VIDEOTRK          JW3       N   Y
LASTNAME     VIDEOTRK           LAST_NAME    EMPLOYEE          JW4       Y   Y
```

Clearing JW2 clears all joins that were issued after JW2 and that use the same host data source. JW1 remains because it was issued prior to JW2, and JW3 remains because it uses a different host data source:

```
JOIN CLEAR JW2
? JOIN
 JOINS CURRENTLY ACTIVE

HOST                            CROSSREFERENCE
FIELD        FILE     TAG       FIELD        FILE     TAG      AS       ALL WH
-----        ----     ---       -----        ----     ---      --       --- --
PRODCODE     VIDEOTRK           PCD          GGSALES           JW1       Y   Y
MOVIECODE    MOVIES             TRANSDATE    VIDEOTRK          JW3       N   Y
```

Listing JOIN Structures: The ? JOIN Query

To display a list of joined data sources, issue the following command at the FOCUS command prompt or in a stored procedure:

```
? JOIN
```

This displays every JOIN command currently in effect. The output now includes a column titled WH for WHERE-based joins:

```
JOINS CURRENTLY ACTIVE
HOST                          CROSSREFERENCE
FIELD        FILE     TAG     FIELD        FILE     TAG      AS    ALL   WH
-----        ----     ---     -----        ----     ---      --    ---   --
TRANSDATE    VIDEOTRK         RELDATE      MOVIES            JW1   N      Y
EMP_ID       EMPLOYEE         EMP_ID       EDUCFILE          J1    Y      N
```

Reference Error Message

(FOC36316) WHERE-BASED JOIN FROM NON-FOCUS TO FOCUS NOT SUPPORTED

This error occurs when a non-FOCUS data source is joined to a FOCUS data source. The feature is supported in FOCUS-to-FOCUS or non-FOCUS-to-non-FOCUS data sources only.

NF767: Long Alphanumeric Fields

You can have up to 3968 bytes in an alpha field within a FOCUS or FUSION file segment. You can have up to 4095 bytes in a fixed format sequential data source. You may define the length in either the Master File, a DEFINE FILE command, or a COMPUTE command.

Long alpha fields are useful for:

- Extracting data from other DBMS's in order to load into a data warehouse.
- Handling long text without the need to use TXT fields.
- Its manipulating capabilities in EDIT, including concatenation, SUBSTR, and CONTAINS.

Example How to Use an Alpha Field Over 256 Bytes

In a Master File:

```
FILENAME=VIDEOTRK, SUFFIX=FOC
SEGNAME=CUST,  SEGTYPE=S1,
 FIELDNAME=CUSTID,CIN,A3860,$
```

In a DEFINE FILE:

```
DEFINE FILE VIDEOTRK
CUSTID/A3860=...;
```

Reference Usage Notes for 4K Alpha fields

- Long alpha fields cannot be indexed.
- For FOCUS and FUSION files, a segment still has to fit on a 4K page. Thus, the maximum length of an alpha field is dependent on the length of the other fields within its segment.

- Long alpha fields cannot be used in a CRTFORM.
- You can print or hold long alpha fields but are unable to view them online.
- Long alpha fields may be used as keys.
- Long alpha fields are not supported in Hot Screen.

NF787: Increased Report Width

The logical size of the internal (FOCSORT) sort page has been increased from approximately 4K to approximately 32K. The sort page is increased in 4K increments, as needed. This allows the width of a result in a TABLE request to approach 32K, depending on the specific request syntax.

In conjunction with the increased sort page size, the maximum number of display fields has increased to 1024 subject to the limitation that the length of all fields in the request be less than the logical sort page size. Each field generates a small amount of overhead that slightly reduces the total length available.

External sorts cannot handle keys that extend past the first 4K page of the sort record. Therefore, using FOCSORT may produce a performance improvement when the keys extend beyond 4K. In this case, FOCSORT is used even if you have EXTSORT set ON.

The increased logical sort page size applies to the TABLE and TABLEF commands. MATCH continues to operate under the previous rules.

NF788: Tiles

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

Grouping Data in Tiles

Grouping data in tiles means sorting the data instances on a BY field in the request and then apportioning them as equally as possible into the number of tile groups you specify. Certain rules are followed when applying this process:

- 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 changes value.
- Instances are counted using the tile field. If the request prints fields from lower level segments, there may be multiple report lines that correspond to one instance of the tile field.

- Instances with the same tile field value are placed in the same tile. For example, consider the following data, which is to be apportioned into three tiles:

  1
  5
  5
  5
  8
  9

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

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

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

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

Syntax How to Group Data In Tiles

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

where:

HIGHEST

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

LOWEST

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

k

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

tilefield

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

head1

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

n

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

m

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

head2

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

Note:

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

Example Grouping Data Into Five Tiles

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

The output is:

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

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

Example Displaying the First Three Tile Groups

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

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

The output is:

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

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

Example Grouping Data Into Tiles and Customizing Column Headings

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

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

The output is:

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

Reference Usage Notes for Tiles

- If a request retrieves data from segments that are descendents of the segment containing the tile field, multiple report rows may correspond to one instance of the tile field. These additional report rows do not affect the number of instances used to assign the tile values. However, if you retrieve fields from multiple segments and create a single-segment extract file, this flat file will have multiple instances of the tile field, and this increased number of instances may affect the tile values assigned. Therefore, when you can run the same request against the multi-level file and the single-segment file different tile assignments may result.
- Tiles are always calculated on a BY sort field in the request.
- Only one tiles calculation is supported per request. However, the request can contain up to five (the maximum allowed) non-tile IN-GROUP-OF phrases in addition to the TILES phrase.

- Comparisons for the purpose of assigning tile numbers use exact data values regardless of their display format. Therefore, if you display a floating-point value as D7, you may not be showing enough significant digits to indicate why values are placed in separate tiles.
- The tile field can be a real field or a virtual field created with a DEFINE command or a DEFINE in the Master File. The COMPUTE command cannot be used to create a tile field.
- Empty tiles do not display in the report output.
- In requests with multiple display commands, tiles are supported only at the lowest level and only with the BY LOWEST phrase.
- Tiles are supported with extract files. However, the field used to calculate the tiles will propagate three fields to a HOLD file unless you set HOLDLIST to PRINTONLY.
- Tiles are not supported with BY TOTAL, TABLEF, FML, and GRAPH.

Reference Tiles Error Messages

```
(FOC32408)   BY HIGHEST/LOWEST n: n should be positive
(FOC32410)   TILES: MORE THAN 1 IN-GROUPS-OF n TILES
(FOC32411)   TILES: n in IN-GROUPS-OF n TILES is too large
(FOC32412)   TILES: IN-GROUPS-OF n TILES TOP m - wrong values
             m and n in request are required to be positive
(FOC32413)   TILES are not supported with BY TOTAL
(FOC32414)   TILES are supported only in TABLE request
```

NF789: FORECAST

The FORECAST feature allows you to uncover trends in numeric data. Depending on the options you specify, it can also provide predicted values beyond the range of the values stored in the data source.

The methods available for calculating trend values are:

- Simple moving average (MOVAVE). This method calculates a series of arithmetic means using a user-specified number of values from a report column.
- Exponential moving average (EXPAVE). This method calculates a weighted average between the previously calculated value of the average and the next data point.
- Linear regression analysis (REGRESS). This method derives the coefficients of a straight line that best fits the data points and uses this linear equation to estimate values.

To generate predicted values, FORECAST continues the same calculations beyond the data points by using the generated trend values as new data points. For the REGRESS technique, the calculated regression equation is used to derive trend and predicted values.

FORECAST Processing

You invoke FORECAST using a special version of the ON *sortfield* RECAP command. In this command you specify the parameters needed for generating estimated values, including the field to be used in the calculations, the method to use, and the number of predictions to generate. T

he RECAP field can be a new field or it can be the same field used in the FORECAST calculations:

- If the RECAP field is the same as the field being used to generate the FORECAST calculations, it is referred to as a *recursive* FORECAST. In this case, the original field is not printed, even if it was referenced in the display command, and the RECAP column contains the original field values followed by the number of predicted values specified in the FORECAST syntax. No trend values display in the report. However, the original column will be propagated to an output file unless you set HOLDLIST to PRINTONLY
- If the RECAP field is a new field, the original field and the new field both display in the report output (if the original field was mentioned in the display command). This is referred to as a *non-recursive* FORECAST. The new field will contain trend values (estimated values *within* the range of the existing data points) and, depending on the arguments you supply, forecast values (predictions *beyond* the range of the existing data points).

The sort field used for FORECAST must be a numeric or smart date field. FORECAST operates on the last ACROSS field in the request. If the request contains no ACROSS fields, it operates on the last BY field. However, to use an ACROSS field with FORECAST the display command must be SUM (or its synonyms ADD or WRITE) or COUNT. The command cannot be PRINT or LIST. The FORECAST calculations start over when the highest-level sort field changes its value. In a request with multiple display commands, FORECAST operates on the last ACROSS field (or if there are no ACROSS fields, the last BY field) of the last display command.

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

Syntax How to Use FORECAST

The following syntax is for the MOVAVE and EXPAVE methods:

```
ON sfld RECAP fld1[/fmt] = FORECAST(fld2, interval, npredict,
'method',npoint);
```

The following syntax is for the REGRESS method (omits the *npoint* parameter):

```
ON sfld RECAP fld1[/fmt] = FORECAST(fld2, interval, npredict, 'REGRESS');
```

where:

sfld

Is the last ACROSS field in the request and must be a numeric or smart date field. If the request contains no ACROSS phrases, FORECAST works on the last BY field. However, FORECAST is only supported with ACROSS when the display command is SUM, WRITE, ADD, or COUNT.

fld1

Is a numeric field. It can be a real field, a virtual field, or a calculated field.

Note: The word FORECAST and the opening parenthesis must be on the same line as the syntax *fld1*=.

fmt

Is the display format for *fld1*. If it is omitted, the default format is D12.2. Even if *fld1* was previously reformatted using a DEFINE or COMPUTE command, the format specified in the RECAP command is respected.

`fld2`

Can be any numeric field. If it is the same as *fld1* (recursive), the predicted values will be appended to the report column after the data values, and the original column is not printed in the report. If it is a different name than *fld1* (non-recursive), this new column will be calculated containing both trend values (estimated values within the range of the existing data points) and, if you specify a non-zero number of predictions, forecast values (predictions beyond the range of the existing data points), while retaining the original field as a separate report column, if it was referenced in the display command.

`interval`

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

For a date sort field, the minimal component in the format determines how the number is interpreted. For example, if the format is YMD, MDY, or DMY, an interval value of 2 is interpreted as meaning two days; if the format is YM, the 2 is interpreted as meaning two months.

`npredict`

Is the number of predictions for FORECAST to calculate. It must be a whole number greater than or equal to zero. Zero indicates that you want no predictions. Zero is only supported when the RECAP field is a new field (non-recursive), not the same field used as the first parameter to FORECAST. If you supply a number that is not a whole number, the fractional portion is dropped.

method

Is the estimation method to use. It can be one of the following values enclosed in single quotation marks:

Method	Definition
MOVAVE	Simple moving average
EXPAVE	Exponentially smoothed moving average
REGRESS	Linear regression

npoint

Is a positive whole number that specifies the number of values to average for the MOVAVE method. For EXPAVE, this number is used to calculate the weights for each component in the average. This parameter must be specified for MOVAVE and EXPAVE and omitted for REGRESS. If you supply a number that is not a whole number, the fractional portion is dropped.

Reference Usage Notes for FORECAST

- For averages, data values should be evenly spaced in order to get meaningful results.
- The RECAP command used with FORECAST can contain only the FORECAST syntax. FORECAST does not recognize any syntax after the closing semicolon (;). To specify options such as AS or IN:
  - If it a non-recursive FORECAST request (creates a new field), use an empty COMPUTE command prior to the RECAP.
  - If it is a recursive FORECAST request, specify the options when the field is first referenced in the report request.

- FORECAST operates on the last ACROSS field, and if the request contains no ACROSS phrases, FORECAST operates on the last BY field.
- FORECAST is only supported for ACROSS fields if the display command is SUM, COUNT, WRITE, or ADD. You cannot use FORECAST on an ACROSS field if the display command is LIST or PRINT.
- In a request with multiple display commands, FORECAST must be applied to the last ACROSS field (or if there are no ACROSS fields, the last BY field) in the last display command. If you use FORECAST to recalculate a field in the request, the original value of the field will be used everywhere except in the columns displayed by the last display command in the request.
- BY TOTAL is not supported.
- MORE, MATCH, FOR, and OVER are not supported.
- The LINES and RECORDS statistics are affected by FORECAST.
- The process of generating the FORECAST values creates extra columns that are not printed in the report output. The number and placement of these additional columns varies depending on the specific request. Therefore, use of column notation is not supported in a request that includes FORECAST.
- SUMMARIZE and RECOMPUTE are not supported for the same sort field used for FORECAST.
- FORECAST is not supported for the FOCUS GRAPH facility; it is supported for the iWay GRAPH facility.
- A request can contain up to seven non-FORECAST RECAP commands and up to seven additional FORECAST RECAP commands.

- The left side of a RECAP command used for FORECAST supports the CURR attribute for doing currency conversions. The MISSING attribute is not supported in the RECAP command.
- A request that creates a new field using the REGRESS method with the SUM command produces a regression based on the detail data values, not the summed values. This is not true if the REGRESS method is used to replace an existing field.

Forecasting Methods

The methods available with FORECAST may sometimes be used to predict values outside the range of the existing data points. However, these methods are not always reliable predictors. Many factors determine how accurate a prediction will be. The FORECAST operation performs the calculations based on the data provided. Decisions about their use and reliability are the user's responsibility.

Using a Simple Moving Average

A simple moving average is a series of arithmetic means calculated with a user-specified number of values, n, from a report column. Each new mean in the series is calculated by dropping the first value used in the prior calculation and adding the next data value to the calculation.

Simple moving averages are sometimes used to analyze trends in stock prices over time. In this scenario, the average is calculated using n periods worth of stock prices. A disadvantage to this indicator is that because it drops the oldest values from the calculation as it moves on in time, it loses its memory over time. Also, mean values are distorted by extreme highs and lows and give equal weight to each point.

Predicted values beyond the range of the data values are calculated using a moving average that treats the calculated trend values as new data points.

The first complete moving average occurs at the nth 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 nth row at which point there are enough values to calculate the moving average with the number of values specified.

Example Calculating a New Simple Moving Average Column

This request defines an integer value named PERIOD to use as the independent variable for the moving average. It predicts three periods of values beyond the range of the retrieved data:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 SMONTH/M = SDATE;
 PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
  SUM UNITS DOLLARS
  BY  CATEGORY BY PERIOD
  WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
  ON PERIOD RECAP MOVAVE/D10.1= FORECAST(DOLLARS,1,3,'MOVAVE',3);
END
```

The output is:

Category	PERIOD	Unit Sales	Dollar Sales	MOVAVE
Coffee	1	61666	801123	801,123.0
	2	54870	682340	741,731.5
	3	61608	765078	749,513.7
	4	57050	691274	712,897.3
	5	59229	720444	725,598.7
	6	58466	742457	718,058.3
	7	60771	747253	736,718.0
	8	54633	655896	715,202.0
	9	57829	730327	711,158.7
	10	57012	724412	703,545.0
	11	51110	620264	691,667.7
	12	58981	762328	702,334.7
	13	0	0	694,975.6
	14	0	0	719,879.4
	15	0	0	705,729.9
Food	1	54394	672727	672,727.0
	2	54894	699073	685,900.0
	3	52713	642802	671,534.0
	4	58026	718514	686,796.3
	5	53289	660740	674,018.7
	6	58742	734705	704,653.0
	7	60127	760586	718,677.0
	8	55622	695235	730,175.3
	9	55787	683140	712,987.0
	10	57340	713768	697,381.0
	11	57459	710139	702,349.0
	12	57290	705315	709,740.7
	13	0	0	708,398.2
	14	0	0	707,818.0
	15	0	0	708,652.3

Note:

- The number of values to use in the average is 3.
- Three predicted values of MOVAVE are calculated within each value of CATEGORY. For values outside the range of the data, new PERIOD values are generated by adding the interval value (1) to the prior PERIOD value
- There are no UNITS or DOLLARS values for the generated PERIOD values.
- Each average (MOVAVE value) is computed using DOLLARS values where they exist. For predicted values beyond those points, the calculated MOVAVE values are used as new data points to continue the moving average, PERIOD is the independent variable (x) and MOVAVE is the dependent variable (y).

  The first MOVAVE value (801,123.0) is equal to the first DOLLARS value.

  The second MOVAVE value (741,731.5) is the mean of DOLLARS values one and two: (801123 + 682340) /2.

  The third MOVAVE value (749,513.7) is the mean of DOLLARS values one through three: (801123 + 682340 + 765078) / 3.

  The fourth MOVAVE value (712,897.3) is the mean of DOLLARS values two through four: (682340 + 765078 + 691274) /3.

  The predicted MOVAVE values (starting with 694,975.6 for PERIOD 13) are calculated using the previous MOVAVE values as new data points. For example, the first predicted value (694,975.6) is the average of the data points from periods 11 and 12 (620,264 and 762328) and the moving average for period 12 (702334.7). The calculation is: 694,975 = (620,264 + 762328 + 702334.7)/3.

Example Using an Existing Field as a Simple Moving Average Column

The following is the request in *Calculating a New Simple Moving Average Column*, but using same name for the RECAP field as for the first argument in the FORECAST parameter list. The trend values do not display in the report. The actual data values for DOLLARS are followed by the predicted values in the report column:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 SMONTH/M = SDATE;
 PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
  SUM UNITS DOLLARS
  BY  CATEGORY BY PERIOD
  WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
  ON PERIOD RECAP DOLLARS/D10.1 = FORECAST(
                DOLLARS,1,3,'MOVAVE',3);
END
```

The output is:

Category	PERIOD	Unit Sales	DOLLARS
Coffee	1	61666	801,123.0
	2	54870	682,340.0
	3	61608	765,078.0
	4	57050	691,274.0
	5	59229	720,444.0
	6	58466	742,457.0
	7	60771	747,253.0
	8	54633	655,896.0
	9	57829	730,327.0
	10	57012	724,412.0
	11	51110	620,264.0
	12	58981	762,328.0
	13	0	694,975.6
	14	0	719,879.4
	15	0	705,729.9
Food	1	54394	672,727.0
	2	54894	699,073.0
	3	52713	642,802.0
	4	58026	718,514.0
	5	53289	660,740.0
	6	58742	734,705.0
	7	60127	760,586.0
	8	55622	695,235.0
	9	55787	683,140.0
	10	57340	713,768.0
	11	57459	710,139.0
	12	57290	705,315.0
	13	0	708,398.2
	14	0	707,818.0
	15	0	708,652.3

Example Using a Date Sort Field With a Simple Moving Average

This request defines a smart date field in the form YYMD to use as the independent variable for the moving average. The format YYMD is used so that the interval between predictions will be based on months. The request predicts three months of values beyond the range of the retrieved data. The trend values, predicted values, and analysis are the same as in *Calculating a New Simple Moving Average Column*:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
END
TABLE FILE GGSALES
  SUM UNITS DOLLARS
  BY  CATEGORY BY SDATE
  WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
  ON SDATE RECAP MOVAVE/D10.1= FORECAST(DOLLARS,1,3,'MOVAVE',3);
END
```

The output is:

Category	SDATE	Unit Sales	Dollar Sales	MOVAVE
Coffee	1997/01	61666	801123	801,123.0
	1997/02	54870	682340	741,731.5
	1997/03	61608	765078	749,513.7
	1997/04	57050	691274	712,897.3
	1997/05	59229	720444	725,598.7
	1997/06	58466	742457	718,058.3
	1997/07	60771	747253	736,718.0
	1997/08	54633	655896	715,202.0
	1997/09	57829	730327	711,158.7
	1997/10	57012	724412	703,545.0
	1997/11	51110	620264	691,667.7
	1997/12	58981	762328	702,334.7
	1998/01	0	0	694,975.6
	1998/02	0	0	719,879.4
	1998/03	0	0	705,729.9
Food	1997/01	54394	672727	672,727.0
	1997/02	54894	699073	685,900.0
	1997/03	52713	642802	671,534.0
	1997/04	58026	718514	686,796.3
	1997/05	53289	660740	674,018.7
	1997/06	58742	734705	704,653.0
	1997/07	60127	760586	718,677.0
	1997/08	55622	695235	730,175.3
	1997/09	55787	683140	712,987.0
	1997/10	57340	713768	697,381.0
	1997/11	57459	710139	702,349.0
	1997/12	57290	705315	709,740.7
	1998/01	0	0	708,398.2
	1998/02	0	0	707,818.0
	1998/03	0	0	708,652.3

Using an Exponential Moving Average

This method calculates an average that allows you to choose weights to apply to newer and older values.

The weight given to the newest value is k, where:

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

The quantity n is an integer greater than one. Increasing n increases the weight assigned to the earlier observations (or data instances) as compared to the later ones.

The next calculation of the exponential moving average (EMA) value is derived by the following formula:

```
EMA = (EMA * (1-k)) + (datavalue * k)
```

This means that the newest value from the data source is multiplied by the factor k and the current moving average is multiplied by the factor (1-k). These quantities are then summed to generate the new EMA.

Note: When the data values are exhausted, the last average calculated is used as the next data value, making every predicted value a constant equal to the last calculated average.

Example Calculating a New Exponential Moving Average Column

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 SMONTH/M = SDATE;
 PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
  SUM UNITS DOLLARS
  BY  CATEGORY BY PERIOD
  WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
  ON PERIOD RECAP EXPAVE/D10.1= FORECAST(DOLLARS,1,3,'EXPAVE',3);
END
```

The output is:

Category	PERIOD	Unit Sales	Dollar Sales	EXPAVE
Coffee	1	61666	801123	801,123.0
	2	54870	682340	741,731.5
	3	61608	765078	753,404.8
	4	57050	691274	722,339.4
	5	59229	720444	721,391.7
	6	58466	742457	731,924.3
	7	60771	747253	739,588.7
	8	54633	655896	697,742.3
	9	57829	730327	714,034.7
	10	57012	724412	719,223.3
	11	51110	620264	669,743.7
	12	58981	762328	716,035.8
	13	0	0	716,035.8
	14	0	0	716,035.8
	15	0	0	716,035.8
Food	1	54394	672727	672,727.0
	2	54894	699073	685,900.0
	3	52713	642802	664,351.0
	4	58026	718514	691,432.5
	5	53289	660740	676,086.3
	6	58742	734705	705,395.6
	7	60127	760586	732,990.8
	8	55622	695235	714,112.9
	9	55787	683140	698,626.5
	10	57340	713768	706,197.2
	11	57459	710139	708,168.1
	12	57290	705315	706,741.6
	13	0	0	706,741.6
	14	0	0	706,741.6
	15	0	0	706,741.6

Note:

- The number n, which is used to calculate the weights is 3.
- Three predicted values of EXPAVE are calculated within each value of CATEGORY. For values outside the range of the data, new PERIOD values are generated by adding the interval value (1) to the prior PERIOD value.
- There are no UNITS or DOLLARS values for the generated PERIOD values.
- Each average is computed using DOLLARS values where they exist. For predicted values beyond those points, the calculated EXPAVE values are used as new data points in the exponential average calculation.

  The first EXPAVE value (801,123.0) is the same as the first DOLLARS value.

  The second EXPAVE value (741,731.5) is calculated as follows. Note that because of rounding and the number of decimal places used, the value derived in this sample calculation varies slightly from the one displayed in the report output:

  ```
  n=3 (number used to calculate weights)

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

  EXPAVE = (EXPAVE*(1-k))+(new-DOLLARS*k) = (801123*0.5) +
  (682340*0.50)
         =  400561.5 + 341170 = 741731.5
  ```

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

  ```
  EXPAVE = (EXPAVE*(1-k))+(new-DOLLARS*k) =
  (741731.5*0.5)+(765078*0.50)
         =  370865.75 + 382539 = 753404.75
  ```

The predicted EXPAVE values (starting with 706,741.6) are calculated using the same formula as used to calculate the trend values. However, an exponential average is always calculated using the previous average and the new data point. Because the previous average is also used as the new data point, the predicted values are always equal to the last trend value. For example, for period 13, the previous average is 706,741.6 and this is also used as the next data point, therefore, the average is calculated as follows:

(706,741.6 * 0.5) + (706,741.6 * 0.5) = 706,741.6

```
EXPAVE = (EXPAVE * (1-k)) + (new-DOLLARS * k) = (706741.6*0.5) +
         (706741.6*0.50) =  353370.8 + 353370.8 = 706741.6
```

Example Using an Existing Field for an Exponential Moving Average

The following is the request in *Calculating a New Exponential Moving Average Column*, but using same name for the RECAP field as for 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,'EXPAVE',3);
END
```

The output is:

Category	PERIOD	Unit Sales	DOLLARS
Coffee	1	61666	801,123.0
	2	54870	682,340.0
	3	61608	765,078.0
	4	57050	691,274.0
	5	59229	720,444.0
	6	58466	742,457.0
	7	60771	747,253.0
	8	54633	655,896.0
	9	57829	730,327.0
	10	57012	724,412.0
	11	51110	620,264.0
	12	58981	762,328.0
	13	0	716,035.8
	14	0	716,035.8
	15	0	716,035.8
Food	1	54394	672,727.0
	2	54894	699,073.0
	3	52713	642,802.0
	4	58026	718,514.0
	5	53289	660,740.0
	6	58742	734,705.0
	7	60127	760,586.0
	8	55622	695,235.0
	9	55787	683,140.0
	10	57340	713,768.0
	11	57459	710,139.0
	12	57290	705,315.0
	13	0	706,741.6
	14	0	706,741.6
	15	0	706,741.6

Using a Linear Regression Equation

This method estimates values by assuming that the dependent variable (y, the new calculated values) and the independent variable (x, the sort field values) are related by the following function, which represents a straight line:

$$y = mx + b$$

The value of m represents the slope of the line, and b represents the y-intercept.

REGRESS uses a technique called Ordinary Least Squares to calculate values for m and b that minimize the sum of the squared differences between the data and the resulting line.

The following formulas show how m and b are calculated. In these formulas, n is the number of data points, the y values are the data values (dependent variable), and the x values are the sort field values (independent variable):

$$m = (\Sigma xy - (\Sigma x * \Sigma y)/n) / (\Sigma x^2 - (\Sigma x)^2/n)$$
$$b = (\Sigma y)/n - (m * (\Sigma x)/n)$$

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:

  ```
  FORMPG = (-0.001323 * DEALER_COST) + 29.32
  ```

  The predicted values are (the values are not exactly as calculated by FORECAST because of rounding, but they show the process of calculating the values):

DEALER_COST	Calculation	FORMPG
15,940	(-0.001323 * 15,940) + 29.32	8.23
16,940	(-0.001323 * 16,940) + 29.32	6.91
17,940	(-0.001323 * 17,940) + 29.32	5.59

Example Using an Existing Field as a Linear Regression Column

The following is the request in *Calculating a New Linear Regression Field*, but using same name for the RECAP field as for the first argument in the FORECAST parameter list. The trend values do not display in the report.

The actual data values for MPG are followed by the predicted values in the report column:

```
TABLE FILE CAR
PRINT MPG
BY DEALER_COST
WHERE MPG NE 0
  ON DEALER_COST RECAP MPG = FORECAST(MPG,1000,3,'REGRESS');
END
```

The output is:

```
 DEALER_COST             MPG
 -----------             ---
       2,886           27.00
       4,292           25.00
       4,631           21.00
       4,915           21.00
       5,063           23.00
       5,660           21.00
                       21.00
       5,800           24.20
       6,000           24.20
       7,427           16.00
       8,300           18.00
       8,400           18.00
      10,000           18.00
      11,000           18.00
      11,194            9.00
      14,940           11.00
      15,940            8.21
      16,940            6.88
      17,940            5.55
```

FORECAST Reporting Techniques

You can use FORECAST multiple times in one request. However, the FORECAST requests must all specify the same sort field, interval, and number of predictions. Only the RECAP field, method, field used to calculate the FORECAST values, and number of points to average can change. If you change any of the other parameters, the new parameters are ignored.

If you want to move a FORECAST column in the report output, use an empty COMPUTE command for the FORECAST field as a placeholder. The data type (I, F, P, D) must be the same in the COMPUTE command and the RECAP command.

To make the report output easier to interpret, you can create a field that indicates whether the FORECAST value in each row is a predicted value. To do this, define a virtual field whose value is always a constant other than zero. Rows in the report output that represent actual records in the data source will display this constant. Rows that represent predicted values will display zero. You can also propagate this field to a HOLD file.

Example Generating Multiple FORECAST Columns in a Request

This example calculates moving averages and exponential averages for both the DOLLARS and BUDDOLLARS fields in the GGSALES data source.

The sort field, interval, and number of predictions are the same for all of the calculations:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 SMONTH/M = SDATE;
 PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
  SUM   DOLLARS AS 'DOLLARS' BUDDOLLARS AS 'BUDGET'
  BY  CATEGORY NOPRINT BY PERIOD AS 'PER'
  WHERE SYEAR EQ 97 AND CATEGORY EQ 'Coffee'
  ON PERIOD RECAP DOLMOVAVE/D10.1= FORECAST(DOLLARS,1,0,'MOVAVE',3);
  ON PERIOD RECAP DOLEXPAVE/D10.1= FORECAST(DOLLARS,1,0,'EXPAVE',4);
  ON PERIOD RECAP BUDMOVAVE/D10.1 = FORECAST(BUDDOLLARS,1,0,'MOVAVE',3);
  ON PERIOD RECAP BUDEXPAVE/D10.1 = FORECAST(BUDDOLLARS,1,0,'EXPAVE',4);
END
```

The output is:

PER	DOLLARS	BUDGET	DOLMOVAVE	DOLEXPAVE	BUDMOVAVE	BUDEXPAVE
1	801123	801375	801,123.0	801,123.0	801,375.0	801,375.0
2	682340	725117	741,731.5	753,609.8	763,246.0	770,871.8
3	765078	810367	749,513.7	758,197.1	778,953.0	786,669.9
4	691274	717688	712,897.3	731,427.8	751,057.3	759,077.1
5	720444	739999	725,598.7	727,034.3	756,018.0	751,445.9
6	742457	742586	718,058.3	733,203.4	733,424.3	747,901.9
7	747253	773146	736,718.0	738,823.2	751,910.3	757,999.6
8	655896	685170	715,202.0	705,652.3	733,634.0	728,867.7
9	730327	753760	711,158.7	715,522.2	737,358.7	738,824.6
10	724412	709397	703,545.0	719,078.1	716,109.0	727,053.6
11	620264	630452	691,667.7	679,552.5	697,869.7	688,413.0
12	762328	718837	702,334.7	712,662.7	686,228.7	700,582.6

Example Moving the FORECAST Column

The following example places the DOLLARS field after the MOVAVE field by using an empty COMPUTE command as a placeholder for the MOVAVE field. Both the COMPUTE command and the RECAP command specify formats for MOVAVE (of the same data type), but the format on the RECAP command takes precedence:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 SMONTH/M = SDATE;
 PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
SUM   UNITS
COMPUTE MOVAVE/D10.2 = ;
DOLLARS
  BY  CATEGORY BY PERIOD
  WHERE SYEAR EQ 97 AND CATEGORY EQ 'Coffee'
  ON PERIOD RECAP MOVAVE/D10.1= FORECAST(DOLLARS,1,3,'MOVAVE',3);
END
```

The output is:

Category	PERIOD	Unit Sales	MOVAVE	Dollar Sales
Coffee	1	61666	738,659.3	801123
	2	54870	734,447.5	682340
	3	61608	749,513.7	765078
	4	57050	712,897.3	691274
	5	59229	725,598.7	720444
	6	58466	718,058.3	742457
	7	60771	736,718.0	747253
	8	54633	715,202.0	655896
	9	57829	711,158.7	730327
	10	57012	703,545.0	724412
	11	51110	691,667.7	620264
	12	58981	702,334.7	762328
	13	0	694,975.6	0
	14	0	719,879.4	0
	15	0	705,729.9	0

Example Distinguishing Data Rows From Predicted Rows

In the following example, the DATA_ROW virtual field has the value 1 for each row in the data source. It has the value zero for the predicted rows.

The PREDICT field is calculated as YES for predicted rows and NO for rows containing data:

```
DEFINE FILE CAR
DATA_ROW/I1 = 1;
END
TABLE FILE CAR
  PRINT DATA_ROW
COMPUTE PREDICT/A3 = IF DATA_ROW EQ 1 THEN 'NO' ELSE 'YES' ;
MPG
BY DEALER_COST
WHERE MPG GE 20
  ON DEALER_COST RECAP FORMPG/D12.2=FORECAST(MPG,1000,3,'REGRESS');
  ON DEALER_COST RECAP MPG        =FORECAST(MPG,1000,3,'REGRESS');
END
```

The output is:

```
DEALER_COST  DATA_ROW  PREDICT              MPG          FORMPG
-----------  --------  -------             ---          ------
      2,886         1  NO                27.00           25.65
      4,292         1  NO                25.00           23.91
      4,631         1  NO                21.00           23.49
      4,915         1  NO                21.00           23.14
      5,063         1  NO                23.00           22.95
      5,660         1  NO                21.00           22.21
                    1  NO                21.00           22.21
      5,800         1  NO                24.20           22.04
      6,000         1  NO                24.20           21.79
      7,000         0  YES               20.56           20.56
      8,000         0  YES               19.32           19.32
      9,000         0  YES               18.08           18.08
```

Reference FORECAST Error Messages

(FOC36320) **FORECAST/REGRESS SYNTAX ERROR**

(FOC36322) **ON FIELD IS NOT LAST BY OR ACROSS FIELD**

FORECAST can only be performed on the last ACROSS field. If ACROSS is not specified then on last BY field.

(FOC36323) **EXPRESSIONS ARE NOT PERMITTED IN PARAMETERS**

In FORECAST/REGRESS, parameters cannot be presented as expressions but only as fields or constants.

(FOC36324) **VERB OBJECT OR INTERVAL CANNOT BE A QUOTED CONSTANT**

All parameters of FORECAST/REGRESS except 'method' should not
contain quotes.

(FOC36325) **FORECAST/REGRESS VERB OBJECT IS NOT RECOGNIZED**

The verb object is not found in either a master file or a prior DEFINE section or a COMPUTE statement.

(FOC36326) **INTERVAL OR NUMBER IS NOT CONVERTED AS ASSUMED TYPE**

In FORECAST, parameter 'interval' must have the type of the field which it increments. Parameters 'npoint', and 'npredict' must be a positive whole number.

(FOC36327) **MATHEMATICAL METHOD IS NOT RECOGNIZED**

In FORECAST, only three method functions can be used in a 'method' parameter. They are 'REGRESS', 'MOVAVE', and 'EXPAVE'.

(FOC36328) INCORRECT USE OF NUMERIC FORECAST PARAMETERS

In FORECAST/REGRESS, a numeric parameter such as 'number', 'npredict', 'interval, or 'npoint' is either non-positive or exceeds an allowed limit.

(FOC36329) INVALID FORECAST REQUEST: PRINT/LIST with ACROSS

It is illegal to use FORECAST together with an ACROSS phrase when the command is LIST or PRINT. Only the SUM command can be used in this case.

(FOC36330) INTERNAL FORECAST/REGRESS ERROR

Unforeseen error has occurred in routines for FORECAST/REGRESS.
Please report the problem to an Information Builders representative.

(FOC36333) INVALID FORECAST/REGRESS REQUEST

'MORE', 'OVER', 'GRAPH', 'MODIFY', 'FOR', and 'BY TOTAL' as well as re-formatting into non-numeric type of field are not permitted in FORECAST/REGRESS.

NF790: ACROSS-TOTAL

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

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

Syntax How to Request ACROSS-TOTAL in a Report

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

where:

sortfield

Is the name of the field being sorted across.

name

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

col1, col2

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

Example Requesting ACROSS-TOTAL in a Report

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

The output is:

```
                    RATING
                     PG      R       TOTAL
                     TITLE   TITLE   TITLE
CATEGORY   COPIES    COUNT   COUNT   COUNT
-------------------------------------
ACTION        14       2       3       5
COMEDY        16       4       1       5
DRAMA          2       0       1       1
FOREIGN        5       2       3       5
MUSICALS       2       1       1       2
MYSTERY       17       2       5       7
SCI/FI         3       0       3       3
```

Reference Usage Notes for ACROSS-TOTAL

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

NF792: SET CDN Command Enhancement

Continental Decimal Notation (CDN) includes the SPACE and QUOTE settings to support countries that require spaces and apostrophes in numbers.

Syntax How to Set Continental Decimal Notation

`SET CDN = cdn`

Possible values are:

OFF

Sets the decimal point as a period and the thousands separator as a comma. OFF is the default.

OFF should be used for the USA, Canada, Mexico, and the United Kingdom.

ON

Sets the decimal point as a comma and the thousands separator as a period.

ON should be used for Germany, Denmark, Italy, Spain, and Brazil.

SPACE

Sets the decimal point as a comma and the thousands separator as a space.

SPACE should be used for France, Norway, Sweden, and Finland.

QUOTE

Sets the decimal point as a comma and the thousands separator as an apostrophe.

QUOTE should be used for Switzerland.

Example How Continental Decimal Notation is Displayed

The following table shows how 1234.56 is displayed depending on the setting of CDN.

OFF	1,234.56
ON	1.234,56
SPACE	1 234,56
QUOTE	1’234,56

NF793: Displaying Fractions

You can use the SET CENT-ZERO command to display fractions in a format that accommodates a country's needs. The setting of CDN determines whether a decimal point or comma displays for fractions.

Syntax How to Display Fractions

```
SET CENT-ZERO = {OFF|ON}
```

where:

`OFF`

Displays fractions with either a decimal point or comma in front of the number depending on how Continental Decimal Notation (CDN) is set. OFF is the default.

`ON`

Displays fractions with a leading zero. The fraction contains either a decimal point or comma depending on the setting of CDN.

Example Displaying of Fractions

The following shows how the number 1234.56 and the fraction.12 are displayed depending on the setting of CENT-ZERO.

CENT-ZERO Setting	CDN Setting	1,234.56	.12
OFF	OFF	1,234.56	,12
	ON	1.234,56	,12
	QUOTE	1’234,56	,12
	SPACE	1 234,56	,12
ON	OFF	1,234.56	0.12
	ON	1.234,56	0,12
	QUOTE	1’234,56	0,12
	SPACE	1 234,56	0,12

NF796: Unlimited Nested -INCLUDE Commands

The Dialogue Manager -INCLUDE command dynamically inserts a stored procedure (FOCEXEC file) into another stored procedure at run time. The inserted file can contain an entire procedure or any part of a procedure. For example, you can use this technique to insert common virtual field definitions or heading text into multiple procedures.

-INCLUDE commands are considered *nested* when a file that is inserted into a procedure using the -INCLUDE command itself contains a -INCLUDE command.

Note: The level of nesting is limited only by the available memory. In prior releases, -INCLUDE commands could be nested up to four levels deep.

Example Using Nested -INCLUDE Commands

Consider the following sequence of files:

INCLUDE1:

```
-DEFAULTS &ECHO = ON
TABLE FILE MOVIES
-INCLUDE INCLUDE2
```

INCLUDE2:

```
HEADING CENTER
"UNLIMITED -INCLUDE COMMANDS"
-INCLUDE INCLUDE3
```

INCLUDE3:

```
PRINT TITLE RATING LISTPR
-INCLUDE INCLUDE4
```

INCLUDE4:

```
BY CATEGORY
-INCLUDE INCLUDE5
```

INCLUDE5:

```
WHERE CATEGORY EQ 'ACTION' OR 'MUSICALS'
-INCLUDE INCLUDE6
```

INCLUDE6:

```
ON CATEGORY SUBTOTAL LISTPR
ON TABLE NOTOTAL
END
```

Now issue the following command:

```
EX INCLUDE1
```

The resulting procedure (displayed by setting the variable &ECHO to ON) is:

```
TABLE FILE MOVIES
HEADING CENTER
"UNLIMITED -INCLUDE COMMANDS"
PRINT TITLE RATING LISTPR
BY CATEGORY
WHERE CATEGORY EQ 'ACTION' OR 'MUSICALS'
ON CATEGORY SUBTOTAL LISTPR
ON TABLE NOTOTAL
END
```

The output is:

```
                    UNLIMITED -INCLUDE COMMANDS
CATEGORY  TITLE                                    RATING  LISTPR
--------  -----                                    ------  ------
ACTION    JAWS                                     PG       19.95
          ROBOCOP                                  R        19.98
          TOTAL RECALL                             R        19.99
          TOP GUN                                  PG       14.95
          RAMBO III                                R        19.95

*TOTAL ACTION
                                                            94.82

MUSICALS  CABARET                                  PG       19.98
          ALL THAT JAZZ                            R        19.98
          CHORUS LINE, A                           PG13     14.98
          FIDDLER ON THE ROOF                      G        29.95

*TOTAL MUSICALS
                                                            84.89
```

Reference Usage Notes for Unlimited Nested -INCLUDE Commands

- Recursive nesting of -INCLUDE or EX commands is not allowed. That is, a file that is inserted using the -INCLUDE command cannot directly or indirectly -INCLUDE or EX itself.

  An indirect recursion would occur if procedure A contained a -INCLUDE command for procedure B and procedure B contained a -INCLUDE or EX command for procedure A.

  Recursion is not supported at any level of nesting and generates the following error message:

  ```
  (FOC36249) RECURSIVE -INCLUDES ARE NOT ALLOWED
  ```

Reference Error Messages for Nested -INCLUDE Commands

(FOC36249) **RECURSIVE -INCLUDES ARE NOT ALLOWED**

The -INCLUDE dialogue manager directive may not include the same file recursively (either directly or indirectly).

NF797: Field-Based Reformatting

With field-based reformatting you can apply different formats to each row in a single report column by using a field to identify the format that applies to each row. For example, you can use this technique to apply the appropriate decimal currency formats when each row represents a different country. For example, Japan uses no decimal places while the United States uses two decimal places.

The field that contains the format specifications can be a real field in the data source or a temporary field created with a DEFINE command, a DEFINE in the Master File, or a COMPUTE command. If it is created with a COMPUTE command, the command must appear in the request prior to using the calculated field for reformatting.

Using Field-Based Formatting

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

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

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

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

- Any date field can be reformatted to any other date format type.
- Any date-time field can be reformatted to any other date-time format.

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

Syntax How to Define and Apply a Format Field

You can define the format field as follows:

- With a DEFINE command:

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

- In a Master File:

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

- In a request:

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

where:

format_field

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

expression

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

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

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

where:

display

Is any valid display command.

fieldname

Is a field in the request to be reformatted.

format_field

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

just

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

Example Displaying Different Decimal Places for Currency Values

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

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

The output is:

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

Reference Usage Notes for Field-Based Reformatting

- Field-based reformatting is supported for TABLE and TABLEF. It works with StyleSheets, joins, and for any type of data source.
- Field-based reformatting is not supported for MODIFY, Maintain, MATCH, GRAPH, RECAP, FOOTING, HEADING, or text fields.
- Although you can use a DEFINE or COMPUTE command to create the format field, you *cannot* apply a field-based format to a calculated or virtual field.
- Field-based reformatting cannot be used on a BY sort field. It does work with an ACROSS field.
- If a report column is produced using field-based reformatting, the format used for a total or subtotal of the column will be taken from the previous detail line.
- Explicit reformatting creates two display fields internally for each field that is reformatted. Field-based reformatting creates three display fields.

- Field-based reformatting works for alphanumeric fields in a HOLD file, although three fields will be stored in the file for each field that is reformatted. To prevent the extra fields from being propagated to the HOLD file, specify SET HOLDLIST=PRINTONLY.
- If the number of decimal places varies between rows, the decimal points will not be aligned in the report output.

NF801: DB2 Data Adapter SET SSID Command for MSO

If the CAF option for the DB2 Data Adapter is installed at your site, you must indicate which DB2 subsystem you intend to use. The name of the DB2 subsystem may differ from the default, or your site may have multiple copies of DB2. To specify the DB2 subsystem-ID (SSID), issue the SET SSID command before executing a request. This command is supported for DB2 Version 5 and higher.

Syntax How to Dynamically Change the DB2 Subsystem ID

`SQL [DB2] SET SSID ssid`

where:

ssid

Is the DB2 subsystem ID; the default is DSN, unless your site changed the default at installation time. Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.

To reset the SSID to the installation default, issue the command with a blank *ssid* value.

Reference Usage Notes for the SQL DB2 SET SSID Command

- The subsystem ID remains in effect for the duration of the session unless it is issued from a global profile or overridden by a subsequent SET SSID command.
- The plan associated with the MSO Data Adapter for DB2 must be bound to *all* DB2 subsystems that will be referenced.
- The DB2 Data Adapter must be installed with the CAF option.

- The data adapter may have been installed with a default, site-specific, SSID setting. Use the SQL ? query command to display this setting.
- The data adapter does not check whether the SSID you specified in the command is valid. If you specify an invalid SSID, DB2 will return an error, which will then generate the following messages:

  (FOC1400) SQLCODE IS %1 (HEX: %2) %3%4

  This message indicates an error at execution of an SQL operation. It contains the SQL return code returned by the RDBMS and is usually accompanied by appropriate FOCUS error messages. For additional information about SQL return codes, refer to the RDBMS manual.

  (FOC1456) CAF: SPECIFIED DB2 SSID NOT FOUND %1%2%3%4

  During an attempt to allocate a thread to DB2, the most recently specified subsystem name could not be located. Determine the correct subsystem name, and use the SQL DB2 SET SSID command to change the current setting. If a nonexistent subsystem is the default, the Interface should be reinstalled.

Example Dynamically Changing the DB2 Subsystem ID

```
SQL DB2 SET SSID DSNX
```

Example Querying the DB2 Data Adapter Settings

```
SQL DB2 ?
```

Information about all of the data adapter settings is returned. You may have to page down (using PF8 on MVS) to see every option.

The following line of output shows the Subsystem ID setting:

```
(FOC1447) SSID FOR CALL ATTACH IS             -  : DSNX
```

Example Restoring the DB2 Subsystem ID to the Installation Default

```
SQL DB2 SET SSID
```

NF802: Long Master File Names

Master File names for FOCUS, relational, and fixed format sequential data sources can be up to 64 characters long on OS/390. Except where noted, this length is supported in all functional areas that reference a Master File.

Master Files with long names can also be created by issuing the HOLD command with an AS phrase that specifies a long name.

Using Long Names on OS/390

In the OS/390 environment, file and member names are limited to eight characters. Therefore, longer Master File names are assigned eight-character names to be used when interacting with the operating system. Three new enhancements have been implemented in order to enable this feature on OS/390:

- A LONGNAME option for the DYNAM ALLOCATE command, which creates the long Master File name and performs the allocation. This DYNAM option is described in *How to Allocate a Long Master File Name in OS/390*.
- An eight-character naming convention for member names associated with long Master File names. This convention is described in *Member Names for Long Master File Names in OS/390*.
- A long Master File attribute, $ VIRT, which contains the long name to be used when interacting with the Master File and the operating system. This attribute is described in *How a Long Master File Name is Implemented in OS/390*.

Member Names for Long Master File Names in OS/390

The DYNAM ALLOC command with the LONGNAME option automatically creates a member for the long Master File name in the PDS allocated to ddname HOLDMAST.

The member name consists of three parts: a prefix consisting of the leftmost characters from the long name, followed by a left brace character ({), followed by an index number. This naming convention is in effect for all long Master Files allocated using DYNAM or created using the HOLD command. The length of the prefix depends on how many long names have a common set of leftmost characters:

- The first ten names that share six or more leftmost characters have a six-character prefix and a one-character index number, starting from zero.
- Starting with the eleventh long name that shares the same leftmost six characters, the prefix becomes five characters, and the index number becomes two characters, starting from 00.

This process can continue until the prefix is one character and the index number is six characters. If you delete one of these members from the HOLDMAST PDS, the member name will be reused for the next new long name with the same prefix.

Example Long Master File Names and Corresponding Member Names

The following table lists sample long names with the corresponding member names that would be assigned under OS/390.

Long Name	Member Name
EMPLOYEES_ACCOUNTING	EMPLOY{0
EMPLOYEES_DEVELOPMENT	EMPLOY{1
EMPLOYEES_DISTRIBUTION	EMPLOY{2
EMPLOYEES_FINANCE	EMPLOY{3
EMPLOYEES_INTERNATIONAL	EMPLOY{4
EMPLOYEES_MARKETING	EMPLOY{5
EMPLOYEES_OPERATIONS	EMPLOY{6
EMPLOYEES_PERSONNEL	EMPLOY{7
EMPLOYEES_PUBLICATIONS	EMPLOY{8
EMPLOYEES_RESEARCH	EMPLOY{9
EMPLOYEES_SALES	EMPLO{00
EMPLOYEES_SUPPORT	EMPLO{01

Syntax How a Long Master File Name is Implemented in OS/390

To relate the short name to its corresponding long name, the first line of a long Master File contains the following attribute:

```
$ VIRT=long_filename
```

where:

long_filename

Is the long name, up to 64 characters.

Syntax How to Allocate a Long Master File Name in OS/390

```
DYNAM ALLOC DD ddname LONGNAME long_filename DS physical_filename
```

where:

ddname

Is the one- to eight-character member name of the Master File. It must be an existing member of a PDS allocated to DD MASTER.

long_filename

Is the long Master File name. The DYNAM command creates a copy of the short Master File in the PDS allocated to DD HOLDMAST. The member in HOLDMAST conforms to the eight character naming convention for long names. The Master File has the $ VIRT attribute on the top line, which contains the long name.

Note: The copy, not the member ddname, is the Master File used when you reference the long name in a request.

physical_filename

Is the data set name of the FOCUS or fixed format sequential data source.

After you have allocated the long name, you can reference the data source using the long Master File name or the short ddname.

Syntax How to Free an Allocation for a Long Master File Name

```
DYNAM FREE LONGNAME long_filename
```

where:

long_filename

Is the long Master File name.

After issuing the DYNAM FREE LONGNAME command, you can no longer reference the data source using the long Master File name. However, you can reference it using the short ddname that was specified in the DYNAM ALLOC command.

Example Using a Long Master File Name on OS/390

To reference the EMPLOYEE data source as EMPLOYEE_DATA, dynamically allocate the long name:

```
DYNAM ALLOC DD EMPLOYEE LONGNAME EMPLOYEE_DATA -
  DS USER1.EMPLOYEE.FOCUS SHR REU
```

You can now issue a request using the long name:

```
TABLE FILE EMPLOYEE_DATA
PRINT CURR_SAL
BY LAST_NAME BY FIRST_NAME
END
```

The output is:

```
LAST_NAME        FIRST_NAME         CURR_SAL
---------        ----------         --------
BANNING          JOHN             $29,710.00
BLACKWOOD        ROSEMARIE        $21,790.00
CROSS            BARBARA          $27,072.00
GREENSPAN        MARY              $9,010.00
IRVING           JOAN             $26,872.00
JONES            DIANE            $18,490.00
MCCOY            JOHN             $18,490.00
MCKNIGHT         ROGER            $16,110.00
ROMANS           ANTHONY          $21,130.00
SMITH            MARY             $13,210.00
                 RICHARD           $9,510.00
STEVENS          ALFRED           $11,010.00
```

In this example, the long Master File will exist in the HOLDMAST PDS as member EMPLOY{0. The index number after the bracket depends on the number of existing long Master Files containing the same first six leftmost characters. The content of the EMPLOYEE_DATA Master File is virtually identical to the short Master File used in the allocation. The only difference is the $ VIRT keyword on line one, which contains the long name. The FILENAME parameter also contains the long name, up to 64 characters.

```
$ VIRT=EMPLOYEE_DATA
$ Created from EMPLOYEE      MASTER
FILENAME=EMPLOYEE_DATA,
SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,       ALIAS=EID,     FORMAT=A9,       $
 FIELDNAME=LAST_NAME,    ALIAS=LN,      FORMAT=A15,      $
          .
          .
          .
```

Creating a HOLD Master File With a Long Name on OS/390

If you use the HOLD command to create a FOCUS or fixed format sequential data source with a long name, a long member name will be generated for the accompanying Master File. As with the DYNAM command, the long member name will conform to the eight character naming convention for long file names. To relate the long name to the short member name, the $ VIRT attribute will be generated on the top line in the Master File.

Note: The resulting HOLD file will be a temporary data file. To allocate the long Master File name to a permanent data file, issue the DYNAM command prior to the HOLD request. The ddname in the command must refer to an existing member of the MASTER PDS.

Creating RDBMS Tables With Long Master File Names

You can use the HOLD command to create an RDBMS table if you have the appropriate Write data adapter installed and have authority to create tables. For example, with the DB2 Data Adapter, you can issue the HOLD FORMAT DB2 command.

If the HOLD command includes a long AS name, both the Master and Access File names will be long. On OS/390, the Master and Access Files will be named according to the procedure described in *Using Long Names on OS/390.* The short names assigned to corresponding Master and Access Files will be the same.

The AS name also becomes the table name and is specified in the Access File. If the AS name specified in the HOLD command is longer than the table name length supported by the RDBMS, the table cannot be created.

Reference Maximum Name Lengths for RDBMS Tables

The following table shows maximum name length supported by each RDBMS:

RDBMS	Maximum Table Name Length
DB2	18
Oracle 7, 8, 8i	30
Teradata	30

Example Creating a DB2 Table With a Long Name

The following request creates a 15 character DB2 table named EMPLOYEEINFODB2 (18 is the maximum table name length for DB2):

```
TABLE FILE EMPLOYEE
PRINT EMP_ID CURR_SAL
BY DEPARTMENT
BY LAST_NAME
BY FIRST_NAME
ON TABLE HOLD AS USER1.EMPLOYEEINFODB2 FORMAT DB2
END
```

This request creates the following Master File:

```
$ VIRT=EMPLOYEEINFODB2

FILE=EMPLOYEEINFODB2        ,SUFFIX=SQL
SEGNAME=SEG01    ,SEGTYPE=S0
FIELDNAME    ='DEPARTMENT'        ,'DPT'          ,A10       ,A10       ,$
FIELDNAME    ='LAST_NAME'         ,'LN'           ,A15       ,A15       ,$
FIELDNAME    ='FIRST_NAME'        ,'FN'           ,A10       ,A10       ,$
FIELDNAME    =FOCLIST             ,FOCLIST        ,I5        ,I4        ,$
FIELDNAME    ='EMP_ID'            ,'EID'          ,A9        ,A9        ,$
FIELDNAME    ='CURR_SAL'          ,'CSAL'         ,D12.2M    ,D8        ,$
```

This request also creates the following Access File:

```
$ VIRT=EMPLOYEEINFODB2
SEGNAME=SEG01    ,
TABLENAME=USER1.EMPLOYEEINFODB2
KEYS=04   , WRITE=YES, $
```

Note that the AS name is also the table name.

Consider what happens when the AS name is longer than the 18 characters supported for table names in DB2 on OS/390:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID CURR_SAL
BY DEPARTMENT
BY LAST_NAME
BY FIRST_NAME
ON TABLE HOLD AS EMPLOYEEINFORMATIONDB2 FORMAT DB2
END
```

Running this request generates the following messages:

```
>   NUMBER OF RECORDS IN TABLE=       12  LINES=     12

 HOLDING SQL      FILE...
(FOC1400) SQLCODE IS -107 (HEX: FFFFFF95)
(FOC1414) EXECUTE IMMEDIATE ERROR.
```

In this example, the Master and Access Files are created, however, the table cannot be created because the table name specified in the Access File is too long and, therefore, invalid. Note that you will get a more descriptive message if you issue the following command:

```
SQL DB2 SET ERRORTYPE DBMS
```

For example:

```
(FOC1400) SQLCODE IS -107 (HEX: FFFFFF95)
  :  DSNT408I SQLCODE = -107, ERROR:  THE NAME EMPLOYEEINFORMATIONDB2 IS
TOO
  :          LONG.  MAXIMUM ALLOWABLE SIZE IS 18
  :  DSNT418I SQLSTATE   = 42622 SQLSTATE RETURN CODE
  :  DSNT415I SQLERRP    = DSNHSMUD SQL PROCEDURE DETECTING ERROR
  :  DSNT416I SQLERRD    = 0  0  0  -1  15  0 SQL DIAGNOSTIC INFORMATION
  :  DSNT416I SQLERRD    = X'00000000'  X'00000000'  X'00000000'
  :          X'FFFFFFFF'  X'0000000F'  X'00000000' SQL DIAGNOSTIC
  :           INFORMATION
(FOC1414) EXECUTE IMMEDIATE ERROR.
```

Reference Usage Notes for Long Master File Names

- The FOCUS Database Server (SU) is not supported on any platform.
- The DATASET attribute is not supported in a long Master File.
- The ACCESSFILE attribute is not supported with long Master Files.

- An External index is not supported.
- The LONGNAME option of the DYNAM command may only be issued from within a FOCEXEC or RPC. It cannot be used to pre-allocate long Master Files in JCL or CLISTS on OS/390.
- Long Master Files are not designed to be edited on OS/390. Each time the DYNAM command is issued with the LONGNAME attribute; it overlays the existing member in HOLDMAST. You must make any edits (such as the addition of fields or DBA attributes, or use of the REBUILD utility) to an existing short Master File.
- ? FDT and ? FILE *longfilename* will show an internal DD alias of @000000*n*, where *n* is less than or equal to the number of existing long file allocations. Use this internal DDNAME in all queries that require a valid DDNAME, such as ? TSO DDNAME queries or USE commands (OS/390 only).
- VM is not supported.
- Fusion is not supported.

Reference Error Messages for Long Master File Names

```
(FOC1982) UNABLE TO FIND LONGNAME %1
```

NF817: Creating Comma-Delimited Files

Formats COMMA, COM and COMT give you a choice of comma-delimited file formats for your report output. A comma-delimited file is a sequential data source in which data values are separated by commas. In the output file, all character values are enclosed in double quotation marks. In addition, you can choose to:

- Remove leading blanks from numeric fields and trailing blanks from alphanumeric fields, so that only significant data is propagated to the output file.
- Propagate only the data values to the output file or include the column headings as the first record in the output file.

Single or double quotation marks are supported in character fields. Double quotation marks within a character value are stored as two consecutive double quotation marks.

Types of Comma-Delimited Output Files

The differences between format COMMA, COM, and COMT output files are summarized in the following table:

Format Name	Remove non-significant blanks?	Create Master File?	Suffix in Master File	Extension or Filetype
COMMA	No	No	N/A	PRN
COM	Yes	Yes	COM	CSV
COMT	Yes	Yes	COMT	CSV

Note:

- Format COMMA 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.
- Format COM 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.
- Format COMT 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.

A Master File is created for format COM and COMT when the command used to create the output file is HOLD. The SUFFIX in the generated Master File is COM or COMT.

All values are stored in their actual formats. The following display options are propagated to the output file: leading zeros, zero suppression, and scientific notation. Master file formats generated are exactly as in FORMAT ALPHA.

Syntax How to Save Report Output in Comma-Delimited Format

```
[ON TABLE] command [AS filename] FORMAT comma_format
```

where:

`ON TABLE`

Is the syntax for creating an output file within a TABLE request.

`command`

Can be one of the following:

HOLD creates a Master File to accompany a FORMAT COM or COMT output file. For format COMMA, no accompanying Master File is created.

PCHOLD is used to automatically send the output file to a client application when issued in an environment that supports automatic file transfers.

SAVE creates an output file with no accompanying Master File.

`filename`

Is the file name of the resulting output file on VM, UNIX, or NT, or the ddname on OS/390.

`comma_format`

COMMA creates a comma-delimited variable-length text file containing the data values from the report request. All character values are enclosed in double quotation marks. Corresponding columns in each record are the same length. On VM, the filetype is PRN. On UNIX and Windows NT, the extension is .prn. **Note:** No Master File is created to accompany this output file.

COM creates a comma-delimited variable-length text file containing the data values from the report request. All character values are enclosed in double quotation marks. Leading blanks are stripped from numeric fields (format codes D, F, I, or P) and trailing blanks are stripped from character fields (format code A). Corresponding columns in each record may have different lengths because of the removal of leading and trailing blanks. On VM, the filetype is CSV. On UNIX and Windows NT, the extension is .csv. **Note:** A Master File is created if the HOLD command is used. In the accompanying Master File, the SUFFIX value is COM. The setting PCOMMA=ON is required in order to interpret the character fields correctly in a request. With PCOMMA=OFF (the default), double quotation marks are considered characters within the field, not delimiters around the field.

COMT creates a comma-delimited variable-length text file containing the column headings and data values from the report request. All character values are enclosed in double quotation marks. Leading blanks are stripped from numeric fields (format codes D, F, I, or P) and trailing blanks are stripped from character fields (format code A). Corresponding columns in each record may have different lengths because of the removal of leading and trailing blanks. On VM, the filetype is CSV. On UNIX and Windows NT, the extension is .csv. **Note:** A Master File is created if the HOLD command is used. In the accompanying Master File, the SUFFIX value is COMT.

Reference Usage Notes for Comma-Delimited Output Files

- There is an 8K overhead for processing comma-delimited files using HOLD FORMAT COM or COMT. Therefore, extract files created by HOLD FORMAT COM | COMT are limited to 26K record lengths.

- You must be aware of the formats and lengths supported by the application for which you are creating the output file. Field lengths created by the HOLD or SAVE command are those supported as USAGE formats in a Master File. You must take into account any limits of the destination application.
- The setting PCOMMA=ON is required in order to interpret the character fields created by HOLD FORMAT COM correctly in a request. With PCOMMA=OFF (the default), double quotation marks are considered characters within the field, not delimiters around the field.
- If you use field-based reformatting in the request that creates the output file, the field lengths in the output file are the lengths from the original Master File or DEFINE command.
- If the request that creates the output file retrieves missing data described with MISSING=ON in the original Master File, the output file will contain a blank if the original field was alphanumeric, or a zero if the field was numeric.
- As with all HOLD files, column headings specified with an AS phrase are ignored unless you set ASNAMES to ON. When ASNAMES is set ON, the value from the AS phrase, up to but not including the comma, becomes the field name in the HOLD Master File.
- Regardless of the ASNAMES setting, the column heading in the COMT data file is the value of the AS phrase exactly as specified in the request.
- The SAVB command is not supported. The output file is a text file.
- The OVER phrase is not supported in a request that creates a COMMA, COM, or COMT format output file.

- Smart date fields and dates formatted as I or P fields with date format options are treated as numeric and are not enclosed in double quotation marks in the output file. Dates formatted as alphanumeric fields with date format options are treated as alphanumeric and enclosed in double quotation marks.
- Continental decimal notation (CDN=ON|SPACE|QUOTE) is not supported. A comma within a number would be interpreted as two separate columns by a destination application such as Microsoft Access.
- Text fields are not supported.
- There is a 32K record length limit in the output file
- Blank fieldnames will show up as blank column titles. This may result in an error when attempting to use the file as input to various applications.
- A double-precision number expressed in scientific notation is stored with the character D before the exponent. For example, the command PRINT RETAIL_COST/D9.3E prints the values in the following form: 0.888D+04. Some applications may not understand this value, as they generally expect an E before the exponent. Printing the value as single precision floating point instead of double precision produces a value with an E before the exponent. For example, format F9.3E produces the value 0.888E+04.

Example Creating a Comma-Delimited File With Leading and Trailing Blanks

```
TABLE FILE MOVIES
PRINT COPIES TITLE WHOLESALEPR AS 'WHOLESALE,PRICE'
BY CATEGORY
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
  ON TABLE HOLD AS COMMAH FORMAT COMMA
END
```

Leading blanks are retained in numeric fields and trailing blanks are retained in character fields in the COMMAH output file:

```
"ACTION  ",  2,"JAWS                                ", 10.99
"ACTION  ",  3,"ROBOCOP                            ", 11.50
"ACTION  ",  4,"TOTAL RECALL                       ", 11.99
"ACTION  ",  2,"TOP GUN                            ",  9.99
"ACTION  ",  3,"RAMBO III                          ", 10.99
"CHILDREN",  1,"SMURFS, THE                        ", 10.00
"CHILDREN",  2,"SHAGGY DOG, THE                    ", 29.99
"CHILDREN",  1,"SCOOBY-DOO-A DOG IN THE RUFF       ",  9.75
"CHILDREN",  2,"ALICE IN WONDERLAND                ", 12.50
"CHILDREN",  1,"SESAME STREET-BEDTIME STORIES AND SONGS",  7.65
"CHILDREN",  1,"ROMPER ROOM-ASK MISS MOLLY         ",  7.99
"CHILDREN",  1,"SLEEPING BEAUTY                    ", 15.99
"CHILDREN",  3,"BAMBI                              ", 12.00
```

No Master File is created for use with this output file.

Example Creating a Compressed Comma-Delimited File Without Column Headings

```
TABLE FILE MOVIES
PRINT COPIES TITLE WHOLESALEPR AS 'WHOLESALE,PRICE'
BY CATEGORY
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
  ON TABLE SAVE AS COMH FORMAT COM
END
```

Leading blanks are removed from numeric fields and trailing blanks are removed from character fields in the COMH output file:

```
"ACTION",2,"JAWS",10.99
"ACTION",3,"ROBOCOP",11.50
"ACTION",4,"TOTAL RECALL",11.99
"ACTION",2,"TOP GUN",9.99
"ACTION",3,"RAMBO III",10.99
"CHILDREN",1,"SMURFS, THE",10.00
"CHILDREN",2,"SHAGGY DOG, THE",29.99
"CHILDREN",1,"SCOOBY-DOO-A DOG IN THE RUFF",9.75
"CHILDREN",2,"ALICE IN WONDERLAND",12.50
"CHILDREN",1,"SESAME STREET-BEDTIME STORIES AND SONGS",7.65
"CHILDREN",1,"ROMPER ROOM-ASK MISS MOLLY",7.99
"CHILDREN",1,"SLEEPING BEAUTY",15.99
"CHILDREN",3,"BAMBI",12.00
```

No Master File is created for use with this output file because the SAVE command was used. The HOLD command would generate an accompanying Master File.

Example Creating a Compressed Comma-Delimited File With Column Headings

```
TABLE FILE MOVIES
PRINT COPIES TITLE WHOLESALEPR AS 'WHOLESALE,PRICE'
BY CATEGORY
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
  ON TABLE HOLD AS COMTH FORMAT COMT
END
```

Leading blanks are removed from numeric fields and trailing blanks are removed from character fields in the COMTH output file, and the first row contains column headings. The AS phrase becomes the column heading for the WHOLESALEPR field regardless of the ASNAMES setting:

```
"CATEGORY","COPIES","TITLE","WHOLESALE,PRICE"
"ACTION",2,"JAWS",10.99
"ACTION",3,"ROBOCOP",11.50
"ACTION",4,"TOTAL RECALL",11.99
"ACTION",2,"TOP GUN",9.99
"ACTION",3,"RAMBO III",10.99
"CHILDREN",1,"SMURFS, THE",10.00
"CHILDREN",2,"SHAGGY DOG, THE",29.99
"CHILDREN",1,"SCOOBY-DOO-A DOG IN THE RUFF",9.75
"CHILDREN",2,"ALICE IN WONDERLAND",12.50
"CHILDREN",1,"SESAME STREET-BEDTIME STORIES AND SONGS",7.65
"CHILDREN",1,"ROMPER ROOM-ASK MISS MOLLY",7.99
"CHILDREN",1,"SLEEPING BEAUTY",15.99
"CHILDREN",3,"BAMBI",12.00
```

With ASNAMES set to OFF, the COMTH Master File is:

```
FILE=COMTH          ,SUFFIX=COMT
SEGNAME=COMTH   ,SEGTYPE=S01
FIELDNAME   =CATEGORY           ,E01        ,A8       ,A08      ,$
FIELDNAME   =COPIES             ,E02        ,I3       ,A03      ,$
FIELDNAME   =TITLE              ,E03        ,A39      ,A39      ,$
FIELDNAME   =WHOLESALEPR        ,E04        ,F6.2     ,A06      ,$
```

With ASNAMES set to ON, the COMTH Master File is:

```
FILE=COMTH          ,SUFFIX=COMT
SEGNAME=COMTH   ,SEGTYPE=S01
FIELDNAME   =CATEGORY           ,E01        ,A8       ,A08      ,$
FIELDNAME   =COPIES             ,E02        ,I3       ,A03      ,$
FIELDNAME   =TITLE              ,E03        ,A39      ,A39      ,$
FIELDNAME   =WHOLESALE          ,E04        ,F6.2     ,A06      ,$
```

Example Using Date Formats With HOLD FORMAT COMT

The following request prints the date field RELDATE (format YMD) and computes the following fields:

- IDATE, format I6YMD.
- PDATE, format P6YMD.
- ADATE, format A6YMD.

```
TABLE FILE MOVIES
  PRINT TITLE RELDATE
  COMPUTE
  IDATE/I6YMD = RELDATE;
  PDATE/P6YMD = RELDATE;
  ADATE/A6YMD = RELDATE;
BY CATEGORY
WHERE CATEGORY EQ 'ACTION'
  ON TABLE HOLD AS COMTH FORMAT COMT
END
```

In the output file, only ADATE is enclosed in double quotation marks:

```
"CATEGORY","TITLE","RELDATE","IDATE","PDATE","ADATE"
"ACTION","JAWS",780513,780513,780513,"780513"
"ACTION","ROBOCOP",880603,880603,880603,"880603"
"ACTION","TOTAL RECALL",910324,910324,910324,"910324"
"ACTION","TOP GUN",880124,880124,880124,"880124"
"ACTION","RAMBO III",891216,891216,891216,"891216"
```

Example Opening a Format COMT File in Microsoft Excel

After transferring the COMTH file created in *Creating a Compressed Comma-Delimited File With Column Headings* to the PC using FTP, you can open it in a Microsoft Application such as Excel or Access:

	A	B	C	D
1	CATEGORY	COPIES	TITLE	WHOLESALE,PRICE
2	ACTION	2	JAWS	10.99
3	ACTION	3	ROBOCOP	11.5
4	ACTION	4	TOTAL RECALL	11.99
5	ACTION	2	TOP GUN	9.99
6	ACTION	3	RAMBO III	10.99
7	CHILDREN	1	SMURFS, THE	10
8	CHILDREN	2	SHAGGY DOG, THE	29.99
9	CHILDREN	1	SCOOBY-DOO-A DOG IN THE R	9.75
10	CHILDREN	2	ALICE IN WONDERLAND	12.5
11	CHILDREN	1	SESAME STREET-BEDTIME ST	7.65
12	CHILDREN	1	ROMPER ROOM-ASK MISS MO	7.99
13	CHILDREN	1	SLEEPING BEAUTY	15.99
14	CHILDREN	3	BAMBI	12

Excel recognizes the data as comma-delimited. You may need to size the columns.

NF824: STRIP Subroutine

The STRIP subroutine removes all occurrences of a specific character from an input string. The resulting string has the same length as the original string but is padded on the right with blanks.

Syntax How to Remove All Occurrences of a Character From an Input String

`STRIP(length, source_string, strip_char, result)`

where:

length

Is a number or numeric field that specifies the length of the source and result strings.

source_string

Is an alphanumeric string or field from which the character will be removed. Text fields are not supported.

strip_char

Is an alphanumeric literal or field that specifies the single character to be removed from the source string. If it is a literal, it must be enclosed in single quotation marks. If it is a field, the left-most character in the field will be used as the strip character.

result

Is the name of the alphanumeric output field or its format enclosed in single quotation marks.

Note: In Dialogue Manager, amper variables will have the length specified by the result string.

Example Removing All Occurrences of a Character From a String

The following example removes all occurrences of a period (.) from director names in the comedy category of the MOVIES data source:

```
TABLE FILE MOVIES
PRINT DIRECTOR AND COMPUTE
SDIR/A17 = STRIP(17,DIRECTOR,'.','A17');
WHERE CATEGORY EQ 'COMEDY'
END
```

The output is:

```
DIRECTOR            SDIR
--------            ----
ZEMECKIS R.         ZEMECKIS R
ABRAHAMS J.         ABRAHAMS J
ALLEN W.            ALLEN W
HALLSTROM L.        HALLSTROM L
MARSHALL P.         MARSHALL P
BROOKS J.L.         BROOKS JL
```

Example Removing Single Quotation Marks From a String

To specify a single quotation mark, use two consecutive single quotation marks. You must then enclose this character combination in single quotation marks:

```
TABLE FILE MOVIES
PRINT TITLE AND COMPUTE
STITLE/A39 = STRIP(39,TITLE,'''','A39');
WHERE TITLE CONTAINS ''''
END
```

The output is:

```
TITLE                                    STITLE
-----                                    ------
BABETTE'S FEAST                          BABETTES FEAST
JANE FONDA'S COMPLETE WORKOUT            JANE FONDAS COMPLETE WORKOUT
JANE FONDA'S NEW WORKOUT                 JANE FONDAS NEW WORKOUT
MICKEY MANTLE'S BASEBALL TIPS            MICKEY MANTLES BASEBALL TIPS
```

NF825: SET PCOMMA

By default, when a Master File specifies SUFFIX=COM, incoming alphanumeric values are not enclosed in double quotation marks, and each record is terminated with a comma and dollar sign (,$) character combination. This format does not support retrieval of most comma-delimited files produced by PC applications. A comma-delimited file is a sequential data source in which the data values are separated by commas.

Setting PCOMMA to ON indicates that alphanumeric data will be enclosed in double quotation marks and that each record will be terminated with a carriage return and line feed (crlf). This setting supports retrieval of comma-delimited files created by PC applications and by the HOLD FORMAT COM command.

Syntax How to Retrieve PC-Type Comma-Delimited Data Sources

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

or, in a TABLE request

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

where:

`ON`

Supports retrieval of comma-delimited data sources in which alphanumeric data is enclosed in double quotation marks and in which each record is terminated with a carriage return and line feed.

`OFF`

Supports retrieval of comma-delimited data sources in which alphanumeric data is not enclosed in double quotation marks and in which each record is terminated with a comma and dollar sign.

Example Reading a Comma-Delimited Data Source

The following request creates a comma-delimited output file with character values enclosed in double quotation marks:

```
TABLE FILE MOVIES
PRINT COPIES TITLE WHOLESALEPR RELDATE
BY CATEGORY
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
  ON TABLE HOLD AS COMH FORMAT COM
END
```

The contents of the output file are:

```
"ACTION",2,"JAWS",10.99,780513
"ACTION",3,"ROBOCOP",11.50,880603
"ACTION",4,"TOTAL RECALL",11.99,910324
"ACTION",2,"TOP GUN",9.99,880124
"ACTION",3,"RAMBO III",10.99,891216
"CHILDREN",1,"SMURFS, THE",10.00,880216
"CHILDREN",2,"SHAGGY DOG, THE",29.99,590109
"CHILDREN",1,"SCOOBY-DOO-A DOG IN THE RUFF",9.75,880513
"CHILDREN",2,"ALICE IN WONDERLAND",12.50,510721
"CHILDREN",1,"SESAME STREET-BEDTIME STORIES AND SONGS",7.65,870904
"CHILDREN",1,"ROMPER ROOM-ASK MISS MOLLY",7.99,880723
"CHILDREN",1,"SLEEPING BEAUTY",15.99,750830
"CHILDREN",3,"BAMBI",12.00,420703
```

The following is a Master File that describes this output file:

```
FILE=COMH          ,SUFFIX=COM
SEGNAME=COMH   ,SEGTYPE=S01
FIELDNAME   =CATEGORY          ,E01         ,A8       ,A08      ,$
FIELDNAME   =COPIES            ,E02         ,I3       ,A03      ,$
FIELDNAME   =TITLE             ,E03         ,A39      ,A39      ,$
FIELDNAME   =WHOLESALEPR       ,E04         ,F6.2     ,A06      ,$
FIELDNAME   =RELDATE           ,E05         ,YMD      ,A06      ,$
```

The following request reads this comma-delimited output file:

```
SET PCOMMA = ON
TABLE FILE COMH
PRINT *
END
```

The output is:

```
CATEGORY COPIES TITLE                                WHOLESALEPR RELDATE
-------- ------ -----                                ----------- -------
ACTION        2 JAWS                                       10.99 78/05/13
ACTION        3 ROBOCOP                                    11.50 88/06/03
ACTION        4 TOTAL RECALL                               11.99 91/03/24
ACTION        2 TOP GUN                                     9.99 88/01/24
ACTION        3 RAMBO III                                  10.99 89/12/16
CHILDREN      1 SMURFS, THE                                10.00 88/02/16
CHILDREN      2 SHAGGY DOG, THE                            29.99 59/01/09
CHILDREN      1 SCOOBY-DOO-A DOG IN THE RUFF                9.75 88/05/13
CHILDREN      2 ALICE IN WONDERLAND                        12.50 51/07/21
CHILDREN      1 SESAME STREET-BEDTIME STORIES AND SONGS     7.65 87/09/04
CHILDREN      1 ROMPER ROOM-ASK MISS MOLLY                  7.99 88/07/23
CHILDREN      1 SLEEPING BEAUTY                            15.99 75/08/30
CHILDREN      3 BAMBI                                      12.00 42/07/03
```

NF826: SQUEEZ Subroutine

The SQUEEZ subroutine reduces multiple contiguous blank characters within an input string to a single blank character. The resulting string has the same length as the original string but is padded on the right with blanks.

Syntax How to Reduce Multiple Blanks to a Single Blank

`SQUEEZ(length, source_string, result)`

where:

length

Is a number or numeric field that specifies the length of the source and result fields.

source_string

Is an alphanumeric string or field from which the extra blank characters will be removed. Text fields are not supported.

result

Is the name of the alphanumeric output field or its format enclosed in single quotation marks.

Note: In Dialogue Manager, amper variables will have the length specified by the result string.

Example Reducing Multiple Blanks to a Single Blank

The following request concatenates employee first names and last names into a field called NAME, retaining all blanks. It then uses the SQUEEZ subroutine to reduce multiple blanks within the name to a single blank:

```
DEFINE FILE EMPLOYEE
NAME/A30 = FIRST_NAME | LAST_NAME;
END
TABLE FILE EMPLOYEE
PRINT NAME AND COMPUTE
SQNAME/A30 = SQUEEZ(30,NAME,'A30');
WHERE DEPARTMENT EQ 'MIS'
END
```

The output is:

```
NAME                            SQNAME
----                            ------
MARY      SMITH                 MARY SMITH
DIANE     JONES                 DIANE JONES
JOHN      MCCOY                 JOHN MCCOY
ROSEMARIE BLACKWOOD             ROSEMARIE BLACKWOOD
MARY      GREENSPAN             MARY GREENSPAN
BARBARA   CROSS                 BARBARA CROSS
```

NF827: TRIM Subroutine

The TRIM subroutine removes leading and/or trailing occurrences of a pattern within a string.

Syntax How to Remove Leading and Trailing Occurrences

`TRIM(trim_where, string, string_length, pattern, pattern_length, result)`

where:

trim_where

Alphanumeric

Indicates where to remove the pattern. Possible values are:

`'L'` Removes leading occurrences.

`'T'` Removes trailing occurrences.

`'B'` Removes both leading and trailing occurrences.

string

Alphanumeric

Is the source string. Text fields are not supported.

string_length

Integer

Is the length of the source string.

pattern

Alphanumeric

Is the pattern to remove.

pattern_length

Integer

Is the length of the pattern string.

result

Alphanumeric

Is the name of the alphanumeric output field or its format enclosed in single quotation marks.

Example Removing Leading Occurrences

The following request uses the TRIM subroutine to remove leading occurrences of the characters BR from director names in the MOVIES data source.

```
TABLE FILE MOVIES
PRINT  DIRECTOR AND
COMPUTE
  TRIMDIR/A17 = TRIM('L',DIRECTOR,17,'BR',2,'A17');
  WHERE DIRECTOR CONTAINS 'BR'
END
```

The output is:

```
DIRECTOR           TRIMDIR
--------           -------
ABRAHAMS J.        ABRAHAMS J.
BROOKS R.          OOKS R.
BROOKS J.L.        OOKS J.L.
```

Example Removing Trailing Occurrences

The following request removes trailing occurrences of the characters ER from the TITLE field in the MOVIES data source. In order to remove trailing non-blank characters, trailing blanks must be removed first. The TITLE field has trailing blanks. Therefore, the TRIM subroutine does not remove the characters ER when creating field TRIMT. The SHORT field does not have trailing blanks. Therefore, TRIM removes the trailing ER characters when creating field TRIMS:

```
DEFINE FILE MOVIES
SHORT/A19 = SUBSTR(19, TITLE, 1, 19, 19, SHORT);
END
TABLE FILE MOVIES
PRINT  TITLE  IN 1  AS 'TITLE: '
       SHORT  IN 40 AS 'SHORT: ' OVER
COMPUTE
  TRIMT/A39 = TRIM('T',TITLE,39,'ER',2,'A39'); IN 1 AS 'TRIMT: '
COMPUTE
  TRIMS/A19 = TRIM('T',SHORT,19,'ER',2,'A19'); IN 40 AS 'TRIMS: '
WHERE TITLE LIKE '%ER'
END
```

The output is:

```
TITLE:   LEARN TO SKI BETTER          SHORT:   LEARN TO SKI BETTER
TRIMT:   LEARN TO SKI BETTER          TRIMS:   LEARN TO SKI BETT
TITLE:   FANNY AND ALEXANDER          SHORT:   FANNY AND ALEXANDER
TRIMT:   FANNY AND ALEXANDER          TRIMS:   FANNY AND ALEXAND
```

NF828: SUFFIX = COM/COMT/TABT

Master Files for comma and tab-delimited sequential data sources can have SUFFIX values of COM, COMT, or TABT. Comma-delimited data sources are sequential data sources in which field values are separated by commas. Tab-delimited data sources are sequential data sources in which field values are separated by tabs.

Comma-delimited and tab-delimited data sources cannot participate in joins.

Accessing SUFFIX=COM Data Sources

A Master File containing the attribute SUFFIX=COM can be used to access two styles of comma-delimited sequential data sources:

- One style is consistent with prior releases. Character values are not enclosed in double quotation marks, and the comma-dollar sign character combination is the record terminator. With this style of comma-delimited data source, records can span multiple lines. A field that contains a comma as a character must be enclosed within single quotation marks.
- The second style is consistent with the current industry standard for comma-delimited data sources. Character values are enclosed in double quotation marks and the crlf (carriage-return, line-feed) character combination is the record terminator. In addition, each input record must be completely contained on a single input line. A double quotation mark within a field is identified by two consecutive double quotation marks.

  Note that the setting PCOMMA=ON is required in conjunction with the SUFFIX=COM Master File when accessing this type of data source in order to correctly interpret the double quotation marks around character values. Without this setting, the double quotation marks are considered characters within the field, not delimiters enclosing the field values.

Accessing SUFFIX=COMT Data Sources

A Master File containing the attribute SUFFIX=COMT can be used to access comma-delimited sequential data sources in which all of the following conditions are met:

- The first record of the data source contains column titles. This record will be ignored when the data source is accessed in a request.
- Character values are enclosed in double quotation marks. A double quotation mark within a field is identified by two consecutive double quotation marks.
- Each record is completely contained on one line and terminated with the crlf character combination.

Accessing SUFFIX=TABT Data Sources

A Master File containing the attribute SUFFIX=TABT can be used to access tab-delimited sequential data sources in which all of the following conditions are met:

- The first record of the data source contains column titles. This record will be ignored when the data source is accessed in a request.
- Character values are not enclosed in double quotation marks.
- Each record is completely contained on one line and terminated with the crlf character combination.

NF829: AUTODATE for FOCUS Data Sources

Each segment of a FOCUS data source can have a timestamp field that records the date and time of the last change to the segment. This field can have any name, but its USAGE format must be AUTODATE. The field will be populated each time its segment instance is updated. The timestamp is stored as format HYYMDS and can be manipulated for reporting purposes using any of the date-time functions.

Creating and Using the AUTODATE Field

In each segment of a FOCUS data source, you can define a field with USAGE = AUTODATE. The AUTODATE field cannot be part of a key field for the segment. Therefore, if the SEGTYPE is S2, the AUTODATE field cannot be the first or second field defined in the segment.

The AUTODATE format specification is supported only for a real field in the Master File, not in a DEFINE or COMPUTE command or a DEFINE in the Master File. However, you can use a DEFINE or COMPUTE command to manipulate or reformat the value stored in the AUTODATE field.

After adding an AUTODATE field to a segment, you must REBUILD the data source. REBUILD will not timestamp the field. It will not have a value until a segment instance is inserted or updated.

If a user-written procedure updates the AUTODATE field, the user-specified value will be overwritten when the segment instance is written to the data source. No message is generated to inform the user that the value was overwritten.

The AUTODATE field can be indexed; however, it is recommended that you make sure that the index is necessary because of the overhead needed to keep the index up to date each time a segment instance changes.

If you create a HOLD file that contains the AUTODATE field, it will be propagated to the HOLD file as a date-time field with the format HYYMDS.

Syntax How to Define an AUTODATE Field for a Segment

```
FIELDNAME = name, ALIAS = alias, {USAGE|FORMAT} = AUTODATE ,$
```

where:

name

Is any valid field name.

alias

Is any valid alias.

Example Defining an AUTODATE Field

The EMPDATE data source can be created by doing a REBUILD DUMP of the EMPLOYEE data source and a REBUILD LOAD into the EMPDATE data source. The Master File for EMPDATE is the same as the Master File for EMPLOYEE with the FILENAME changed and the DATECHK field added:

```
FILENAME=EMPDATE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,       ALIAS=EID,     FORMAT=A9,       $
 FIELDNAME=DATECHK,      ALIAS=DATE,    USAGE=AUTODATE,  $
 FIELDNAME=LAST_NAME,    ALIAS=LN,      FORMAT=A15,      $
   .
   .
   .
```

To add the timestamp information to EMPDATE, run the following procedure:

```
SET TESTDATE = 20010715
TABLE FILE EMPLOYEE
PRINT EMP_ID CURR_SAL
ON TABLE HOLD
END

MODIFY FILE EMPDATE
FIXFORM FROM HOLD
MATCH EMP_ID
ON MATCH COMPUTE CURR_SAL = CURR_SAL + 10;
ON MATCH UPDATE CURR_SAL
ON NOMATCH REJECT
DATA ON HOLD
END
```

You can then reference the AUTODATE field in a DEFINE or COMPUTE command, or display it using a display command. The following request computes the number of days difference between the date 7/31/2001 and the DATECHK field:

```
DEFINE FILE EMPLOYEE
DATE_NOW/HYYMD = DT(20010731);
DIFF_DAYS/D12.2 =  HDIFF(DATE_NOW, DATECHK, 'DAY', 'D12.2');
END
TABLE FILE EMPDATE
PRINT DATECHK DIFF_DAYS
WHERE LAST_NAME EQ 'BANNING'
END
```

The output is:

```
DATECHK                        DIFF_DAYS
-------                        ---------
2001/07/15 15:10:37           16.00
```

Reference Usage Notes for AUTODATE

- PRINT * and PRINT.SEG.*fld* will print the AUTODATE field.
- To display the AUTODATE field on a CRTFORM, Winform, or in FSCAN, you must explicitly reference the AUTODATE field name in the request. CRTFORM * will not display the field. CRTFORM always treats the AUTODATE field as a display only (D.) field.
- MODIFY FIXFORM and FREEFORM requests capture the system date/time per transaction.
- SU updates the AUTODATE field per segment using the date and time on the FOCUS Database Server.
- Maintain will process AUTODATE fields at COMMIT time.
- DBA is permitted on the AUTODATE field; however, when unrestricted fields in the segment are updated, the system will update the AUTODATE field.
- The AUTODATE field does not support the following attributes: MISSING, ACCEPT, and HELPMESSAGE.

NF833: Creating Tab-Delimited Files

Format TABT enables you to create tab-delimited output files. A tab-delimited file is a sequential data source in which the data values are separated by tabs. You can use the HOLD command to automatically generate an accompanying Master File, or use the SAVE command to generate just the output file.

Format TABT enables you to create tab-delimited output files in which leading blanks are removed from numeric fields and trailing blanks are removed from alphanumeric fields, so that only significant data is propagated to the output file. Column headings are included as the first record in the output file. Character values are *not* enclosed in double quotation marks.

A Master File is created for format TABT when the command used to create the output file is HOLD. The SUFFIX in the generated Master File is TABT.

All values are stored in their actual formats. The following display options are propagated to the output file: leading zeros, zero suppression, and scientific notation. Master file formats generated are exactly as in FORMAT ALPHA.

Syntax How to Save Report Output in Tab-Delimited Format

```
[ON TABLE] command [AS filename] FORMAT TABT
```

where:

`ON TABLE`

Is the syntax for creating an output file within a TABLE request.

`command`

Can be one of the following:

HOLD creates a Master File to accompany the generated output file.

PCHOLD is used to automatically send the output file to a client application when issued in an environment that supports automatic file transfers.

SAVE creates an output file with no accompanying Master File.

filename

Is the file name of the resulting output file on VM, UNIX, or NT, or the ddname on OS/390.

The tab-delimited output file is a variable-length text file containing the column headings and data values from the report request. Leading blanks are stripped from numeric fields (format codes D, F, I, or P) and trailing blanks are stripped from character fields (format code A). Corresponding columns in each record may have different lengths because of the removal of leading and trailing blanks. On VM, the filetype is TAB. On UNIX and Windows NT, the extension is .tab. **Note:** A Master File is created if the HOLD command is used to create the output file. In the accompanying Master File, the SUFFIX value is TABT.

Reference Usage Notes for Tab-Delimited Output Files

- You must be aware of the formats and lengths supported by the application for which you are creating the output file. Field lengths created by the HOLD or SAVE command are those supported as USAGE formats in a Master File. You must take into account any limits of the destination application.
- If you use field-based reformatting in the request that creates the output file, the field lengths in the output file are the lengths from the original Master File or DEFINE command.

- If the request that creates the output file retrieves missing data described with MISSING=ON in the original Master File, the output file will contain a blank if the original field was alphanumeric, or a zero if the field was numeric.
- As with all HOLD files, column headings specified with an AS phrase are ignored unless you set ASNAMES to ON. When ASNAMES is set ON, the value from the AS phrase, up to but not including the comma, becomes the field name in the HOLD Master File.
- Regardless of the ASNAMES setting, the column heading in the TABT data file is the value of the AS phrase exactly as specified in the request.
- The SAVB command is not supported.
- The TABLEF command is not supported.
- The OVER phrase is not supported.
- Text fields are not supported.
- There is a 32K record length limit in the output file
- Blank fieldnames will show up as blank column titles. This may result in an error when attempting to use the file as input to various applications.
- A double-precision number expressed in scientific notation is stored with the character D before the exponent. For example, the command PRINT RETAIL_COST/D9.3E prints the values in the following form: 0.888D+04. Some applications may not understand this value, as they generally expect an E before the exponent. Printing the value as single precision floating point instead of double precision produces a value with an E before the exponent. For example, format F9.3E produces the value 0.888E+04.

Example Creating a Tab-Delimited File

```
TABLE FILE MOVIES
PRINT COPIES TITLE WHOLESALEPR AS 'WHOLESALE,PRICE'
BY CATEGORY
WHERE CATEGORY EQ 'ACTION' OR 'CHILDREN'
  ON TABLE HOLD AS TABTH FORMAT TABT
END
```

Leading blanks are removed from numeric fields and trailing blanks are removed from character fields in the TABTH output file, and the first row contains column headings. The AS phrase becomes the column heading for the WHOLESALEPR field regardless of the ASNAMES setting.

The following shows the generated output file. The right arrows represent the tab characters:

```
CATEGORY →  COPIES   →   TITLE→WHOLESALE,PRICE
ACTION   →   2   →  JAWS→ 10.99
ACTION   →   3   →  ROBOCOP  →  11.50
ACTION   →   4   →  TOTAL RECALL   →   11.99
ACTION   →   2   →  TOP GUN  →  9.99
ACTION   →   3   →  RAMBO III → 10.99
CHILDREN →  1   →  SMURFS, THE→10.00
CHILDREN →  2   →  SHAGGY DOG, THE → 29.99
CHILDREN →  1   →  SCOOBY-DOO-A DOG IN THE RUFF→9.75
CHILDREN →  2   →  ALICE IN WONDERLAND  →  12.50
CHILDREN →  1   →  SESAME STREET-BEDTIME STORIES AND SONGS → 7.65
CHILDREN →  1   →  ROMPER ROOM-ASK MISS MOLLY →  7.99
CHILDREN →  1   →  SLEEPING BEAUTY → 15.99
CHILDREN →  3   →  BAMBI→12.00
```

With ASNAMES set to OFF, the TABTH Master File is:

```
FILE=TABTH          ,SUFFIX=TABT
SEGNAME=TABTH   ,SEGTYPE=S01
FIELDNAME   =CATEGORY           ,E01        ,A8       ,A08      ,$
FIELDNAME   =COPIES             ,E02        ,I3       ,A03      ,$
FIELDNAME   =TITLE              ,E03        ,A39      ,A39      ,$
FIELDNAME   =WHOLESALEPR        ,E04        ,F6.2     ,A06      ,$
```

With ASNAMES set to ON, the TABTH Master File is:

```
FILE=TABTH          ,SUFFIX=TABT
SEGNAME=TABTH   ,SEGTYPE=S01
FIELDNAME   =CATEGORY           ,E01        ,A8       ,A08      ,$
FIELDNAME   =COPIES             ,E02        ,I3       ,A03      ,$
FIELDNAME   =TITLE              ,E03        ,A39      ,A39      ,$
FIELDNAME   =WHOLESALE          ,E04        ,F6.2     ,A06      ,$
```

NF880: Exit on Error

Using the SET ERROROUT command, you can control how a batch FOCUS job step or an EDA server responds to error conditions encountered in a procedure.

The ERROROUT setting is ignored in an interactive session. If you issue the SET ERROROUT command in an interactive setting, the following message is returned:

```
SETTING IGNORED IN INTERACTIVE MODE
```

Controlling Batch Error Processing

With the ON setting, the result is as follows when an error message is generated:

- In a batch FOCUS job, the return code is set to 8 and the job step terminates. The user can then test this return code in subsequent job steps.
- For EDA, the procedure terminates and the error message is returned to the client.

In addition, the following message displays to inform the user why the program terminated:

```
Exiting due to Exit on Error
```

Warning messages do not invoke this behavior. When a warning is generated, processing continues as normal.

With the OFF setting, no return code is set. Depending on the specific message, FOCUS determines whether FOCEXEC processing continues.

Users can check a Dialogue Manager variable such as &FOCERRNUM and issue the following command to terminate FOCUS and set *n* as the return code:

```
-QUIT FOCUS n
```

On VM, if you include the QUEUE 'FIN' command in your batch FOCUS EXEC, and if FOCUS terminates as a result of the ERROROUT setting, the queued FIN command will cause CMS to issue a return code of -3, which will overwrite the ERROROUT return code. If you want to see the return code issued by Exit on Error, you can remove the QUEUE 'FIN' command from the EXEC and include the following command immediately after the 'EXEC FOCUS' command to exit and issue the return code:

```
exit rc
```

Syntax How to Control Batch Error Processing

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

where:

ON

In response to any error in a batch FOCUS job step, sets the return code to 8 and terminates the job step. In response to an error on an EDA server, terminates the procedure and returns the error message to the client. This setting is ignored in an interactive session.

OFF

Does not set a return code or automatically terminate a job step or procedure in response to any error message. This is the default value.

Reference Usage Notes for Exit on Error

- In unique cases, Information Builders may choose to reclassify specific errors or warnings or exempt specific errors from this behavior.
- The SET ERROROUT command can be issued in any supported profile, in the batch input stream, or in a FOCEXEC. It is not supported in a TABLE request, using ON TABLE SET syntax.
- If the user code handles return codes by using a technique such as checking &FOCERRNUM having this feature enabled will cause the program to exit and, therefore, ignore the error handling logic. In this situation, the user should set ERROROUT to OFF in order to have the error handling logic respected.

Example Using Exit on Error on OS/390

Consider the following in-stream request in a batch FOCUS job:

```
//SYSIN    DD *
SET ERROROUT = ON
DYNAM ALLOC DD MYFILE DS USER1.MYFILE.DATA SHR REU
TABLE FILE NOMFD
PRINT *
ON TABLE HOLD AS MYFILE
END
FIN
//*
//COPYIT  EXEC PGM=IEBCOPY,COND=(4,LT),REGION=2M
   .
   .
   .
```

The JES log contains the following messages:

```
 TABLE FILE NOMFD
(FOC205) THE DESCRIPTION CANNOT BE FOUND FOR FILE NAMED: NOMFD
 BYPASSING TO END OF COMMAND
Exiting due to Exit on Error...
```

The FOCUS job step terminates in response to the FOC205 error, and the return code for the job step is set to 8.

The COPYIT job step does not execute because the return code is 8, and the parameter COND=(4,LT) means that it only executes if all preceding job steps terminate with a return code between zero and four.

Example Using Exit on Error on VM

The following EXEC executes FOCUS in batch on VM. It contains the exit rc command to return the FOCUS return code to VM. The ERROUT FOCEXEC sets ERROROUT to ON and issues a request that results in an error condition:

```
ATTACH PMSDJG 191 '(' M
'ATTACH FOCLIB'
'ATTACH F7029907 294 (F '
GLOBAL TXTLIB FUSELIB
QUEUE 'EX ERROUT1'
'EXEC FOCUS'
exit rc
```

The VM Batch status shows the return code of 8:

```
,LISXXC01                    Batch Job List                        VM:Batch
 ---------------------------------------------------------------------------
Command  Jobname  Jobid   Owner   Class  -------- Status: CURRENT -------- PREV
         JOB6296  6296    PMSDJG    A    completed; Return Code: 8          END
```

NF849: Extended Currency Symbol Support

By default, the M and N display options print the currency symbol associated with the code page being used by your system. For example, when using the U. S. code page, the M and N options display the dollar sign ($).

Extended currency symbol format options allow you to display the following currency symbols regardless of the code page used: U. S. dollar, euro, British pound, and Japanese yen.

The M and N options continue to operate as in prior releases.

Extended Currency Symbol Format Options

The extended currency symbol format options consist of two characters: an exclamation point followed by one of the supported upper or lower case letters. An upper case letter displays a floating currency symbol on each detail line. A lower case letter displays a fixed currency symbol to the left of the field on the first detail line of each report page. These options are valid for numeric formats (I, D, F, and P).

Syntax How To Display an Extended Currency Symbol

Use the following character combinations as the final two characters in any numeric display format:

Display Option	Description	Example
!d	Fixed dollar sign	D12.2!d
!D	Floating dollar sign	D12.2!D
!e	Fixed euro symbol	F10.2!e
!E	Floating euro symbol	F10.2!E
!l	Fixed British pound sign	D12.1!l
!L	Floating British pound sign	D12.1!L
!y	Fixed Japanese yen symbol	I9!y
!Y	Floating Japanese yen symbol	I9!Y

Reference Usage Notes for Extended Currency Symbol Support

- Format specifications are limited to a total of eight characters.
- The extended currency option must be the last option in the format.
- The extended currency option cannot be used in the same format specification as M or N.

- In order to display or print the extended currency symbols, you must be sure they are supported by the fonts accessible to your emulation software or printer.
- The symbol displayed by the M and N options is dependent on the code page being used. The !d and !D options always display a dollar sign, regardless of the code page.
- Using a fixed currency symbol places the symbol only on the first line of each report page. If you use field-based reformatting to display multiple currency symbols in one report column, only the symbol associated with the first row will display. In this case, you should use floating currency symbols.
- Lower case letters are transmitted as upper case letters by the terminal I/O procedures. Therefore, the fixed extended currency symbols can only be specified in a FOCEXEC.
- In TSO, when you display report output without HotScreen (SET SCREEN=OFF), by default the extended currency symbols do not display because the terminal I/O procedures translate all terminal output to characters that appear in USA EBCDIC keyboard layouts and code charts. You can change this default behavior with the following command:

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

  where:

  ON

  Does not display extended currency symbols. ON is the default.

  OFF

  Displays extended currency symbols.

Example Displaying Extended Currency Symbols

The following request uses field-based reformatting to display the Japanese yen on the report row that represents Japan, the British pound on the row that represents England, and the euro on the row that represents Italy. Note that the comma inclusion display option (C) in the format for England is specified prior to the currency option:

```
DEFINE FILE CAR
CFORMAT/A8 = DECODE COUNTRY('ENGLAND' 'F12.1C!L' 'JAPAN' 'D12!Y'
            ELSE 'D12.2!E');
END

TABLE FILE CAR
PRINT SALES/CFORMAT DEALER_COST/CFORMAT
BY COUNTRY
  WHERE COUNTRY EQ 'ENGLAND' OR 'JAPAN' OR 'ITALY'
  WHERE SALES GT 0
END
```

The output is:

```
COUNTRY          SALES  DEALER COST
-------          -----  -----------
ENGLAND      £12,000.0    £11,194.0
ITALY       €30,200.00   €16,235.00
JAPAN          ¥78,030       ¥5,512
```

NF886: Reusable MSO/CICS Control Blocks

MSO provides a facility to re-use orphaned control blocks. If you time out or get disconnected, the control block that you are using goes into a pool. If you return under normal conditions you will get your old control block back, as long as it is still available.

If you try and restart your MSO connection and the MSO/CICS environment has run out of control blocks the following message is displayed:

```
FOCUS session ended: Error unknown or no longer available
```

NF898: SQL Support for Date-Time Formats

The relational data adapters and the SQL Translator now support the date-time data type introduced in FOCUS 7.1. This data type accurately describes RDBMS TIME, DATE, and TIMESTAMP columns. The AUTODB2 and AUTOSQL procedures have been updated to create Master Files that take advantage of this support. In addition, AUTODB2 now supports the DDF (Distributed Data Facility) feature, which allows three-part table names in which the first part of the name specifies a location.

Date-Time Support for the Relational Data Adapters

The date-time data type (format H) can be used to represent time, date, and timestamp data types available in many relational data sources.

In a Master File, the USAGE format for a date-time field describes which components to display and various options for displaying them. In Master Files for relational data sources, date-time fields must also have an ACTUAL format that indicates how the date-time value is stored in the data source.

ACTUAL formats supported for relational date-time values are listed in the following table. The corresponding USAGE must be an H format.

RDBMS	RDBMS DATA TYPE	ACTUAL FORMAT
DB2 and SQL/DS	`TIME`	`HHIS`
	`TIMESTAMP`	`HYYMDm`
	`DATE`	`DATE`
Oracle	`DATE`	`HYYMDS`

Reference Using Date-Time Columns in Requests

- These data types are supported for Direct SQL Passthru, TABLE, MODIFY, Maintain, and SQL Translator requests. They are not supported for Static TABLE or MODIFY requests.
- AUTODB2, AUTOSQL, and CREATE FILE support these data types.
- Date-time manipulation handled by the FOCUS date-time functions is not converted to SQL.
- The only optimized selection criteria on date-time columns (criteria passed to the RDBMS in the generated SQL) are relational expressions that are valid in an IF clause. For example, comparison of a date-time column with a date-time literal value is optimized, but comparison of a date-time column with another date-time column is not optimized.

Example Using a Date-Time Column in a DB2 Table

The DB2 table named EMPINFO has the following columns:

- HIRE_DATE, with ACTUAL = DATE.
- HIRE_DATE_TIME, with ACTUAL = HYYMDm.
- HIRE_TIME, with ACTUAL = HHIS.

The EMPINFO Master File follows:

```
FILENAME=EMPINFO ,SUFFIX=SQLDS,$

SEGNAME=EMPINFO  ,SEGTYPE=S0,$
 FIELD=EMP_ID          ,ALIAS=EID          ,USAGE=A9      ,ACTUAL=A9      ,$
 FIELD=LAST_NAME       ,ALIAS=LN           ,USAGE=A15     ,ACTUAL=A15     ,$
 FIELD=FIRST_NAME      ,ALIAS=FN           ,USAGE=A10     ,ACTUAL=A10     ,$
 FIELD=HIRE_DATE       ,ALIAS=HDT          ,USAGE=YMD     ,ACTUAL=DATE    ,$
 FIELD=DEPARTMENT      ,ALIAS=DPT          ,USAGE=A10     ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY ,ALIAS=CSAL          ,USAGE=P9.2    ,ACTUAL=P4      ,$
 FIELD=CURR_JOBCODE    ,ALIAS=CJC          ,USAGE=A3      ,ACTUAL=A3      ,$
 FIELD=ED_HRS          ,ALIAS=OJT          ,USAGE=F6.2    ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN      ,ALIAS=BONUS_PLAN   ,USAGE=I4      ,ACTUAL=I4      ,$
 FIELD=HIRE_DATE_TIME ,ALIAS=HDTT          ,USAGE=HYYMDm ,ACTUAL=HYYMDm ,$
 FIELD=HIRE_TIME       ,ALIAS=HT           ,USAGE=HHIS    ,ACTUAL=HHIS    ,$
```

The following request prints the date-time columns:

```
TABLE FILE EMPINFO
PRINT HIRE_DATE HIRE_DATE_TIME HIRE_TIME
BY LAST_NAME BY FIRST_NAME
END
```

The output is:

```
LAST_NAME       FIRST_NAME HIRE_DATE HIRE_DATE_TIME             HIRE_TIME
---------       ---------- --------- --------------             ---------
BANNING         JOHN       82/08/01  1982/08/01 16:41:29.453858 16:41:29
BLACKWOOD       ROSEMARIE  82/04/01  1982/04/01 08:41:26.584838 08:41:26
CROSS           BARBARA    81/11/02  1981/11/02 12:30:01.398742 12:30:01
GREENSPAN       MARY       82/04/01  1982/04/01 15:30:17.194398 15:30:17
IRVING          JOAN       82/01/04  1982/01/04 15:45:04.394574 15:45:04
JONES           DIANE      82/05/01  1982/05/01 14:43:56.348577 14:43:56
MCCOY           JOHN       81/07/01  1981/07/01 09:39:46.492837 09:39:46
MCKNIGHT        ROGER      82/02/02  1982/02/02 17:36:21.685746 17:36:21
ROMANS          ANTHONY    82/07/01  1982/07/01 07:22:34.784738 07:22:34
SMITH           MARY       81/07/01  1981/07/01 07:36:25.838573 07:36:25
                RICHARD    82/01/04  1982/01/04 08:02:02.994847 08:02:02
STEVENS         ALFRED     80/06/02  1980/06/02 10:35:42.275949 10:35:42
```

AUTODB2 Support for DDF

AutoDB2 now supports retrieval from and Master File creation for tables from secondary locations. When a location is specified, the TABLENAME attribute in the Access File consists of three parts: *location.creator.tablename*.

During installation, a list of valid locations is included in the AUTODB2 utility. The list appears on the main menu, just below the row in which the user can enter the location. The first value in the list appears by default in the main menu. The user can enter any one the values in the list. Entering a value not on the list generates the following error message:

```
PLEASE ENTER A VALID LOCATION.
```

Table, column, and index information is retrieved from the selected location's catalog tables. The location is included in a three-part table name in the Access File if it is not blank.

Note: The menu option 'Use Creator Name in AFD?=N' is ignored when a non-blank location is provided. That is, the creator is included in the three-part table name even when you specify to not include it on the main menu.

The value for location that you include on the menu is logged to the parameter log file when you press PF4 to log parameters. The next session of AutoDB2/SQL will display the logged value.

ProcedureHow to Install AUTODB2 DDF Support

Create a list of valid locations by editing the following lines in the AUTODB2 FOCEXEC. Uncomment the -DEFAULT command and enter up to six location names. The start and end of the list must be enclosed in single quotation marks, and each name must be eight characters long, padded on the right with blanks if necessary:

```
-
*=========================================================================
-* Change &LOC_LIST to contain possible location values, padded to 8
-* characters, separated by commas.  The entire string is enclosed in
-* single quotes.  Include a maximum of 6 locations.  The first entry is
-* the default. Example:
-* -DEFAULT &LOC_LIST='PROD    ,TEST    ,DEVELOPM'
-*                 <------>,<------>,<------>,<------>,<------>,<------>
-DEFAULT &LOC_LIST='        '
```

A blank location name designates the location where the plan was bound.

Example Creating a Location List

With the following list, blank will be the default value that appears on the main menu.

```
-DEFAULT &LOC_LIST='        ,LOCDSNA ,LOCDSNC '
```

Note: Specifying blank (' ') as the first option indicates that the default is the location where the plan was bound.

AUTODB2/SQL Support for TIME and TIMESTAMP

AUTODB2/SQL now supports native RDBMS TIMESTAMP and TIME formats. You can provide any legal format on the main menu for TIMESTAMP and TIME column types. All TIMESTAMP and TIME columns will be described with the format you provide. The default USAGE format for TIMESTAMP is HYYMDm and the default format for TIME is HHIS. Values are verified, and invalid entries generate an error. To provide backward compatibility, the previously used values of A26 (TIMESTAMP) and A8 (TIME) can still be entered.

Note: These formats are case sensitive. You must type in the values on the AUTODB2/SQL menu in the proper case. The values for TIMESTMP and TIME formats that you include on the menu are logged to the parameter log file when you press PF4 to log parameters. The next session of AUTODB2/SQL will display the logged values.

AUTODB2/SQL Screens That Support DDF and Date-Time Formats

The AUTODB2/SQL main menu now includes entries for the location parameter and for TIME and TIMESTAMP formats. (For detailed instructions on entering information on these screens, see the *DB2 and SQL/DS Read/Write Interface User's Manual.*)

The default values display initially. However, you can change these defaults by entering your preferences and logging them to the parameter log file by pressing PF4:

```
Main Menu     Master File Generation Facility for DB2
     Master Filename ================> timestmp

 Location => locdsna   Creator => USER1     Table => TIMESTMP
 Location values:          ,LOCDSNA ,LOCDSNC
        Database Name ================> *
      Description will be a member of:
        Master Target PDS => PMSLCC.MASTER.DATA
        Access Target PDS => PMSLCC.FOCSQL.DATA
        FOCDEF Target PDS => PMSLCC.FOCDEF.DATA
        Replace Existing Description?=> Y       (Y/N)
        Read/Write Functionality =====> W       (R=Read,W=Write)
        Date Display Format ==========> YYMD
        Time Stamp Display Format ====> HYYMDm
        Time Display Format ==========> HHIS
        Display Decimal when SCALE=0?=> Y       (Y/N)
        Use LABEL as Column Heading? => N       (Y/N)
        Use Remarks for FOCDEF? ======> N       (Y/N)
        Use Creator Name in AFD? =====> Y       (Y/N)
        Use Long Fieldnames? =========> Y       (Y/N)
        Parm File => PMSLCC.FOCSQL.DATA

PF1=Help PF2=Restart PF3=Exit PF4=Log PF5=MFD PF6=AFD PF9=Picture PF10=List
```

The table selection screen now also displays the location parameters for the list of tables generated by your selections on the main menu:

```
Master:       Master File Generation Facility for DB2
TIMSTMP                ==Table Selection==

Place an 'R' next to the Table to be the root of the Master.
Place  a 'C' next to all other Tables to be described as children.
Enter 'Y' next to all selected Tables that will be updated.

Location    Creator Name       DB2 Table Name    Select (R/C)  Write (Y/N)
--------    ------------       --------------    ------------  -----------
LOCDSNA      PMSLCC             TIMESTMP               r              Y



PF1=Help             PF3=End  PF4=Add Tables              PF7=Up  PF8=Down
```

When the Master and Access Files are generated, you return to the main menu with the DESCRIPTION CREATED message at the bottom:

```
Main Menu     Master File Generation Facility for DB2
      Master Filename ================> TIMSTMP

  Location => LOCDSNA   Creator => PMSLCC    Table => TIMESTMP
  Location values:          ,LOCDSNA ,LOCDSNC
        Database Name ================> *
      Description will be a member of:
        Master Target PDS => PMSLCC.MASTER.DATA
        Access Target PDS => PMSLCC.FOCSQL.DATA
        FOCDEF Target PDS => PMSLCC.FOCDEF.DATA
        Replace Existing Description?=> Y       (Y/N)
        Read/Write Functionality =====> W       (R=Read,W=Write)
        Date Display Format ==========> YYMD
        Time Stamp Display Format ====> HYYMDm
        Time Display Format ==========> HHIS
        Display Decimal when SCALE=0?=> Y       (Y/N)
        Use LABEL as Column Heading? => N       (Y/N)
        Use Remarks for FOCDEF? ======> N       (Y/N)
        Use Creator Name in AFD? =====> Y       (Y/N)
        Use Long Fieldnames? =========> Y       (Y/N)
        Parm File => PMSLCC.FOCSQL.DATA
  DESCRIPTION CREATED
PF1=Help PF2=Restart PF3=Exit PF4=Log PF5=MFD PF6=AFD PF9=Picture PF10=List
```

At this point, you can press PF5 to open the Master File or PF6 to open the Access File in TED.

The generated Master File follows:

```
$$$ CREATED BY AUTODB2 ON 11/14/01 AT 12.46.37 BY USER1
FILENAME=TIMSTMP,SUFFIX=SQLDS,$

SEGNAME='TIMESTMP',SEGTYPE=S0,$
FIELD=ALPHA8            ,ALPHA8            ,A8      ,A8     ,MISSING=ON,$
FIELD=DATE1             ,DATE1             ,YYMD    ,DATE   ,MISSING=ON,$
FIELD=TIME1             ,TIME1             ,HHIS    ,HHIS   ,MISSING=ON,$
FIELD=TIMESTMP          ,TIMESTMP          ,HYYMDm  ,HYYMDm ,MISSING=ON,$
```

The generated Access File follows:

```
$$$ CREATED BY AUTODB2 ON 11/14/01 AT 12.46.37 BY USER1
$$$ FILENAME=TIMSTMP,SUFFIX=SQLDS,$

SEGNAME='TIMESTMP',TABLENAME='"LOCDSNA"."USER1"."TIMESTMP"',
KEYS=00,WRITE=YES,KEYORDER=LOW,$
```

Executing AUTODB2 in Batch

To execute AUTODB2 in batch and take advantage of DDF and TIMESTMP support, pass values to the FOCEXEC using the following syntax:

```
EX AUTODB2 BATCH=Y,MASTER=master,CREATOR=creator,TABLENAME=table
    [,option1=value1 ...]
```

where:

master

Is the 1- to 8-character name of the Master File that will be generated.

creator

Is the 1- to 8-character name of the table's creator.

table

Is the 1- to 18-character name of the RDBMS table.

option1 ...

Is an option listed in the following chart.

value1 ...

Is an acceptable value for the corresponding option.

The following options are available:

Option Name	Values	Description	Default
LOC	**location**	**Location value**	**First value in the location list**
REPLACE	Y=Yes,N=No	Replace existing description	N
FUNC	R=Read, W=Write	Read/Write Functionality	W
DATEDISP	*format*	Date Display Format	YYMD
TIMESTMP	**format**	**Time Stamp Display Format**	**HYYMDm**
TIME	**format**	**Time Display Format**	**HHIS**
DECIMAL	Y=Yes,N=No	Display Decimal when SCALE=0?	Y
LABELS	Y=Yes,N=No	Use Labels as Column Heading?	N

Option Name	Values	Description	Default
REMARKS	Y=Yes,N=No	Use Remarks for FOCDEF?	N
CREATAFD	Y=Yes,N=No	Use Creator Name in AFD?	Y
LONG	Y=Yes,N=No	Use Long Filednames?	Y
USERID	*userid*	High level qualifier of output data sets (required if target data set names not provided - MVS only)	
MFDLIST	Y=Yes,N=No	Store list of Master Files created per session	N
MASTERDATA	*dsn*	Master Target PDS (MVS only)	&USERID.MASTER.DATA
FOCSQLDATA	*dsn*	Access Target PDS (MVS only)	&USERID.FOCSQL.DATA
FOCDEFDATA	*dsn*	FOCDEF Target PDS (MVS only)	&USERID.FOCDEF.DATA

Note the addition of LOC=, TIMESTMP=, and TIME= to this list. The default for LOC is the first value in LOC_LIST (provided at installation time). If you want a value of blank, use blank as the first value in LOC_LIST or execute AUTODB2 with the following syntax:

```
EX AUTODB2 LOC=' ' .
```

Reference Usage Notes for AUTODB2/SQL

- AUTODB2/SQL is a FOCEXEC that has a corresponding Master File. The FOCEXEC and Master File must be from the same release of FOCUS. If you attempt to use the old version of the Master File with the new version of the FOCEXEC, the following message is generated:

  ```
  PROGRAM AND FILE DESCRIPTION DATES DO NOT MATCH: AUTODB2 date
  UNABLE TO EXECUTE AUTODB2 - PLEASE REINSTALL
  ```

  FOCEXEC execution is terminated, and you return to the FOCUS prompt.
- As of FOCUS 7.0.9, AUTODB2 no longer needs the DB2CAT Master and Access Files and AUTOSQL no longer needs the SYSCOL Master File or INDEXES Access File. These Master and Access Files remain on the distribution tape. They are copied as part of the installation on VM; MVS instructions tell you to copy them also. However, Information Builders no longer certifies the accuracy of these Master and Access Files with future RDBMS catalog table changes.

Reference AUTODB2/SQL Error Messages

Not using the appropriate AUTODB2/SQL Master File generates the following message:

```
PROGRAM AND FILE DESCRIPTION DATES DO NOT MATCH: AUTODB2 date
UNABLE TO EXECUTE AUTODB2 - PLEASE REINSTALL
```

supplying an invalid location value generates the following message:

```
PLEASE ENTER A VALID LOCATION
```

Supplying an invalid USAGE format for TIMESTAMP generates the following message:

```
INVALID TIME STAMP DISPLAY FORMAT
```

Supplying an invalid USAGE format for TIME generates the following message:

```
INVALID TIME DISPLAY FORMAT
```

SQL Translator Support for Date-Time Fields

Several new data types have been defined to the SQL Translator to support date-time fields in the WHERE predicate or field list of a SELECT statement. In the following discussion, 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	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`

Notice that:

- Time information may be given to the hour, minute, second, millisecond, or microsecond.
- The Translator automatically converts date-time literals to the format required by the RDBMS. Therefore, when you specify a date-time literal, you can use a hyphen or slash between the date components, a colon or period between the time components, and a hyphen or space to separate the date from the time.

Operations on Date-Time Data Supported by the SQL Translator

Time or timestamp columns can be defined in relational data sources, and date-time fields can be defined in FOCUS data sources. These are accessible to the translator. Values can be entered using INSERT and UPDATE statements and displayed in SELECT statements.

Time or timestamp data items (columns or literals) can be compared in conditions. Time values or timestamp values can be added or subtracted from each other, with the result being the number of seconds difference. Expressions of the form T + 2 DAYS or TS + 5 YEARS are allowed. These expressions will be translated to calls to the FOCUS date-time functions.

Extracting Date-Time Components Using the SQL Translator

The SQL Translator supports several functions that return components from date-time values. You can also use the EXRACT statement to extract components.

Syntax Date-Time Functions Accepted by the Translator

The following functions return date-time components as integer values. Assume x is a date-time value:

Function	Return value
`YEAR(`x`)`	year
`MONTH(`x`)`	month number
`DAY(`x`)`	day number
`HOUR(`x`)`	hour
`MINUTE(`x`)`	minute
`SECOND(`x`)`	second
`MILLISECOND(`x`)`	millisecond
`MICROSECOND(`x`)`	microsecond

Example Using SQL Translator Date-Time 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 Functions in a SELECT Statement

Using the DB2 table EMPINFO, consider the following SQL Translator request that prints the field HIRE_DATE_TIME, the year value, month value, minute value, and second value:

```
SQL
SELECT HIRE_DATE_TIME,
YEAR(HIRE_DATE_TIME), MONTH(HIRE_DATE_TIME),
MINUTE(HIRE_DATE_TIME), SECOND(HIRE_DATE_TIME)
FROM EMPINFO;
ECHO ON
END
```

The Translator produces the following Direct SQL Passthru request:

```
SQL SQLDS
SELECT SQLCOR01.HDTT , YEAR (SQLCOR01.HDTT ), MONTH (SQLCOR01.HDTT ),
MINUTE (SQLCOR01.HDTT ), SECOND (SQLCOR01.HDTT ) FROM USER1."EMPINFO"
SQLCOR01
 FOR FETCH ONLY;
END
```

The following report output is generated:

HDTT	E02	E03	E04	E05
1980-06-02-10.35.42.275949	1980	6	35	42
1981-07-01-07.36.25.838573	1981	7	36	25
1982-05-01-14.43.56.348577	1982	5	43	56
1982-01-04-08.02.02.994847	1982	1	2	2
1982-08-01-16.41.29.453858	1982	8	41	29
1982-01-04-15.45.04.394574	1982	1	45	4
1982-07-01-07.22.34.784738	1982	7	22	34
1981-07-01-09.39.46.492837	1981	7	39	46
1982-04-01-08.41.26.584838	1982	4	41	26
1982-02-02-17.36.21.685746	1982	2	36	21
1982-04-01-15.30.17.194398	1982	4	30	17
1981-11-02-12.30.01.398742	1981	11	30	1

Assume a FOCUS data source called EMPDT has the same fields and formats as the EMPINFO table. Running the same SQL Translator request against the FOCUS data source produces the following FOCUS request:

```
DEFINE FILE EMPDT
SQLDEF01/I5 = HPART(HIRE_DATE_TIME,'YEAR','I5');
SQLDEF02/I5 = INT(SQLDEF01);
SQLDEF03/I3 = HPART(HIRE_DATE_TIME,'MONTH','I3');
SQLDEF04/I3 = INT(SQLDEF03);
SQLDEF05/I3 = HPART(HIRE_DATE_TIME,'MINUTE','I3');
SQLDEF06/I3 = HPART(HIRE_DATE_TIME,'SECOND','I3');
END
TABLEF FILE EMPDT
PRINT HIRE_DATE_TIME SQLDEF02 SQLDEF04 SQLDEF05 SQLDEF06
ON TABLE SET CARTESIAN ON
ON TABLE SET ASNAMES ON
ON TABLE SET HOLDLIST PRINTONLY
END
```

The report output is the same as that produced using the EMPINFO table. Note that the SQL Translator functions were converted to DB2 functions in the request against the DB2 table and were converted to calls to the date-time functions when run against the FOCUS data source.

Syntax Using the SQL Translator EXTRACT Function to Extract Date-Time Components

You can use the following ANSI standard function to extract date-time components as integer values:

```
EXTRACT(component FROM value)
```

where:

component

Is one of the following: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, MILLISECOND, or MICROSECOND.

value

Is a date-time, DATE, TIME, or TIMESTAMP field, constant or expression.

For example, the following are equivalent:

```
EXTRACT(YEAR FROM TS)
YEAR(TS)
```

Example Using the EXTRACT Function

```
SELECT TS, EXTRACT(YEAR FROM TS), EXTRACT(MONTH FROM TS),
EXTRACT(DAY FROM TS) FROM TBL1
```

This request produces rows similar to the following:

1999-01-01	1999	1	1
2000-03-03	2000	3	3

NF914: LMF Site Code Warning Message

Each site is required to register its site code as part of the FOCUS installation process. If this installation step is not completed, users will see the following warning message every time they enter FOCUS:

```
*************************************************************
* VIOLATION:                                                *
* THE SITE CODE FOR YOUR COMPANY WAS NOT REGISTERED         *
* PROPERLY DURING THE FOCUS INSTALLATION PROCESS.           *
* PLEASE CONTACT IBI. "EX READMEF" FOR COMPLETE DETAILS     *
*************************************************************
```

Note that all LMF (License Management Facility) site code messages are warning messages.

7.1 New Features

General Enhancements

NF696: Calling Subroutines Written in REXX
NF750: DATASET in a Master File
NF751: Date-Time Data Type
NF773: Token Delimited Files
NF778: Dialogue Manager TRUNCATE Function
NF779: FOCUS CRTFORM HTML Translation

Reporting Enhancements

NF692: Aggregating and Sorting Report Columns
NF731: Reporting From Independent Paths
NF749: HOLD FORMAT INTERNAL
NF755: Using FILEDEF for Creating Extract Files
NF761: Comma Suppress Edit Format Option
NF762: Percent Edit Format Option
NF766: DEFINE Functions
NF781: Embedding Text Fields in Headings

Raised Limits

NF777: Two-Gigabyte FOCUS Database Support
NF759: Increased Number of Display Fields

Performance Enhancements

NF777: Partitioned FOCUS Data Sources

Oracle Data Adapter

NF782: Oracle Data Adapter IXSPACE Setting

Adabas Data Adapter

NF785: The Adabas Write Data Adapter for FOCUS

NF692: Aggregating and Sorting Report Columns

Aggregation and sorting may be applied simultaneously to numeric columns in your report in one pass of the data.

Syntax How To Sort by a Report Column

The syntax is:

```
BY [HIGHEST|LOWEST {n} ] TOTAL display field
```

where:

n

Is the number of instances you wish to extract from the data source.

display field

Can be a fieldname, prefixoperator.fieldname or calculated value.

Example Sorting by a Report Column

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

```
TABLE FILE MOVIES
 SUM LISTPR
 BY CATEGORY
 BY RATING
 BY HIGHEST 5 TOTAL AVE.WHOLESALEPR AS 'AVE,WHOLESALEPR'
 PRINT WHOLESALEPR
 BY CATEGORY
 BY RATING
 WHERE CATEGORY EQ 'CLASSIC' OR 'MYSTERY'
END
```

The output is:

```
                   AVE
 CATEGORY  RATING  WHOLESALEPR  LISTPR  WHOLESALEPR
 --------  ------  -----------  ------  -----------
 CLASSIC   G             40.99   89.95        40.99
           NR            16.08  314.76        14.99
                                              20.00
                                              15.99
                                              10.95
                                              10.95
                                               9.99
                                              40.99
                                              10.95
                                              10.99
                                              15.00
 MYSTERY   NR             9.00   19.98         9.00
           PG             9.00   39.96         9.00
                                               9.00
           PG13           9.00   19.98         9.00
           R             16.19  155.89        15.99
```

Reference Requirements

You must have an aggregating display command (SUM) for BY TOTAL to work correctly. A non-aggregating display command (PRINT) simply retrieves the data without aggregating it. The records will be sorted in either ascending or descending sequence based on your query.

NF696: Calling Subroutines Written in REXX

A FOCUS request can now call user-written subroutines coded in REXX. These routines, also called FUSREXX macros, add a 4GL option to the languages supported for user-written subroutines.

Using REXX Subroutines

REXX subroutines are supported in the VM/CMS and MVS environments:

- In CMS, a FUSREXX macro can contain either REXX source code or compiled REXX code created by running the source code through the REXX compiler. In addition, you can load either type of FUSREXX macro into memory using the EXECLOAD command. The compilation and load process reduces the CPU requirements and increases speed. Compilation also is a security tool, making private information difficult to read.
- In MVS, FOCUS supports source versions of REXX subroutines only.

Because of CPU requirements, the use of FUSREXX routines in large production jobs should be monitored carefully.

The following notes apply to the examples in this document:

- REXX versions are not necessarily the same in all operating environments. Therefore, some of the examples may use REXX functions that are not available in your environment.
- The REXX code is listed, but not fully explained. See your REXX documentation for information about REXX instructions and functions.

Syntax How to Call a REXX User-Written Subroutine

In a DEFINE FILE command:

```
DEFINE FILE filename
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
END
```

In a DEFINE attribute in the Master File:

```
DEFINE fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen,
outparm);
```

In a COMPUTE command:

```
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
```

In a Dialogue Manager -SET command:

```
-SET &var = subname(inlen1, inparm1, ..., outlen, outparm);
```

where:

fieldname

Is the name of the field to receive the return value.

An|In

Is the format of the field to receive return value.

subname

Is the name of the REXX routine.

`inlen1, inparm1 ...`

Are the input parameters. Each parameter consists of a pair of values: a length and an alphanumeric parameter value. You can supply the name of an alphanumeric field, an alphanumeric literal, or an expression that resolves to an alphanumeric value. Up to 13 input parameter pairs are supported by FOCUS. Each parameter value can be up to 256 bytes long. **Note:** Dialogue Manager converts input parameters that consist of numeric digits to decimal format, regardless of their original data type. Therefore, you cannot pass numeric input parameters to a REXX routine using -SET.

`outlen, outparm`

Is the output parameter pair, consisting of a length and a return value. In most cases, the return value should be alphanumeric, but integer return values are also supported. The return value can be the name of the field or Dialogue Manager variable to which the value is returned or its USAGE format enclosed in single quotation marks. The return value can be a minimum of one byte long and a maximum (for an alphanumeric value) of 256 bytes. **Note:** If the value returned is integer, *outlen* must be 4 because FOCUS reserves four bytes for integer fields.

`&var`

Is the name of the Dialogue Manager variable to receive the return value.

REXX subroutines:

- Require input data to be character and should return character output. Integer return values are also supported, but the output length in the subroutine call must be four. FOCUS has a 256-byte limit on character variables. This limit also applies to FUSREXX routines. FUSREXX routines return variable length data. For this reason, you must supply the length of the input arguments and the maximum length of the output data.

- Do *not* require any input parameters, but *do* require one return parameter, which *must* return at least one byte of data. It is possible for a FUSREXX function to need no input, such as a function that returns USERID.
- Do not support floating-point numbers (REXX does not have native floating-point conversion routines). All numeric fields should be converted to character format with no commas using a FOCUS function such as EDIT before being passed to the FUSREXX routine. This prevents FOCUS from converting numbers to floating point before passing them to the FUSREXX routine.
- Are not supported in Dialogue Manager -CMS RUN commands.
- On VM/CMS, the FILETYPE of REXX user-written functions is FUSREXX; they can be stored on any accessed disk.
- On MVS, DDNAME FUSREXX must be allocated to a PDS, and that library will be searched before other MVS libraries.
- The search order for subroutines is:
  1. FUSREXX
  2. Standard CMS or MVS search order.

Example Returning the Day of the Week

The FUSREXX routine DOW returns the day of the week an employee was hired. The routine passes one input parameter pair and one return field pair.

```
   DEFINE FILE EMPLOYEE
1. AHDT/A6 = EDIT(HIRE_DATE) ;
2. DAY_OF_WEEK/A9 WITH AHDT= DOW(6,AHDT,9,DAY_OF_WEEK) ;
   END
```

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME HIRE_DATE DAY_OF_WEEK
END
```

1. The input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The return field is up to nine bytes long and is named DAY_OF_WEEK.

The output is:

```
LAST_NAME         HIRE_DATE  DAY_OF_WEEK
---------         ---------  -----------
STEVENS            80/06/02  Monday
SMITH              81/07/01  Wednesday
JONES              82/05/01  Saturday
SMITH              82/01/04  Monday
BANNING            82/08/01  Sunday
IRVING             82/01/04  Monday
ROMANS             82/07/01  Thursday
MCCOY              81/07/01  Wednesday
BLACKWOOD          82/04/01  Thursday
MCKNIGHT           82/02/02  Tuesday
GREENSPAN          82/04/01  Thursday
CROSS              81/11/02  Monday
```

The FUSREXX macro is displayed below. The FUSREXX routine reads the input date, reformats it to MM/DD/YY format, and returns the day of the week using a REXX DATE call.

```
/* DOW routine. Return WEEKDAY from YYMMDD format date */
Arg ymd .
Return Date('W',Translate('34/56/12',ymd,'123456'),'U')
```

Example Returning Text Format

The REXX function called in this request returns the number of copies of each classic movie in text format. It passes one input parameter and one return field.

```
    TABLE FILE MOVIES
    PRINT TITLE AND COMPUTE
1.  ACOPIES/A3 = EDIT(COPIES); AS 'COPIES'
    AND COMPUTE
2.  TXTCOPIES/A8 = NUMCNT(3,ACOPIES,8,TXTCOPIES);
    WHERE CATEGORY EQ 'CLASSIC'
    END
```

1. The input field is 3 bytes long. Data is passed in field ACOPIES. The COPIES field is converted to an alphanumeric field.
2. The return field is up to 8 bytes long and is named TXTCOPIES.

The output is:

```
TITLE                                    COPIES  TXTCOPIES
-----                                    ------  ---------
EAST OF EDEN                             001     One
CITIZEN KANE                             003     Three
CYRANO DE BERGERAC                       001     One
MARTY                                    001     One
MALTESE FALCON, THE                      002     Two
GONE WITH THE WIND                       003     Three
ON THE WATERFRONT                        002     Two
MUTINY ON THE BOUNTY                     002     Two
PHILADELPHIA STORY, THE                  002     Two
CAT ON A HOT TIN ROOF                    002     Two
CASABLANCA                               002     Two
```

The FUSREXX macro is:

```
/* NUMCNT routine. Pass a number from 0 to 10, return a character value */
Arg numbr .
data = 'Zero One Two Three Four Five Six Seven Eight Nine Ten'
numbr = numbr + 1           /* so 0 equals 1 element in array */
Return Word(data,numbr)
```

Example Passing Multiple Arguments

The following example shows how to pass multiple arguments to a FUSREXX routine. It is an interest calculation using the present salary for the employee and the employee start date to calculate a present value. It passes four input parameters and one return field.

```
    DEFINE FILE EMPLOYEE
1.  AHDT/A6     = EDIT(HIRE_DATE) ;
2.  ACSAL/A12   = EDIT(CURR_SAL) ;
3.  DCSAL/D12.2 = CURR_SAL ;
4.  PV/A12      = INTEREST(6,AHDT,6,'&YMD',3,'6.5',12,ACSAL,12,PV) ;
    END

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE DCSAL PV
END
```

1. The first input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The current salary is converted to an alphanumeric field for use in the interest calculation.
3. The current salary is converted to a double-precision field to include commas and a decimal point in the output.

4. The second input field is six bytes long. Data is passed as a FOCUS character variable &YMD in YYMMDD format.

   The third input field is a character value of 6.5, which is 3 bytes long to account for the decimal point in the character string.

   The fourth input field is 12 bytes long. This passes the character field ACSAL.

   The return field is up to 12 bytes long and is named PV.

The output is:

```
LAST_NAME        FIRST_NAME  HIRE_DATE           DCSAL  PV
---------        ----------  ---------           -----  --
STEVENS          ALFRED       80/06/02       11,000.00      14055.14
SMITH            MARY         81/07/01       13,200.00      15939.99
JONES            DIANE        82/05/01       18,480.00      21315.54
SMITH            RICHARD      82/01/04        9,500.00      11155.60
BANNING          JOHN         82/08/01       29,700.00      33770.53
IRVING           JOAN         82/01/04       26,862.00      31543.35
ROMANS           ANTHONY      82/07/01       21,120.00      24131.19
MCCOY            JOHN         81/07/01       18,480.00      22315.99
BLACKWOOD        ROSEMARIE    82/04/01       21,780.00      25238.25
MCKNIGHT         ROGER        82/02/02       16,100.00      18822.66
GREENSPAN        MARY         82/04/01        9,000.00      10429.03
CROSS            BARBARA      81/11/02       27,062.00      32081.82
```

The FUSREXX macro is displayed below. The REXX format command is used to format the return value.

```
/* Simple INTEREST program. dates are yymmdd format  */
Arg start_date,now_date,percent,open_balance, .
begin = Date('B',Translate('34/56/12',start_date,'123456'),'U')
stop  = Date('B',Translate('34/56/12',now_date,'123456'),'U')
valnow = open_balance * (((stop - begin) * (percent / 100)) / 365)
Return Format(valnow,9,2)
```

Example Accepting Multiple Tokens in Parameters

FUSREXX routines can accept multiple tokens in a parameter. The following procedure passes employee information (pay date and monthly gross pay) as separate tokens in the first parameter. It passes three input parameters and one return field.

```
    DEFINE FILE EMPLOYEE
1.  COMPID/A256 = FN | ' ' | LN | ' ' |  DPT | ' ' | EID ;
2.  APD/A6 = EDIT(PAY_DATE) ;
3.  APAY/A12 = EDIT(MO_PAY) ;
4.  OK4RAISE/A1 = OK4RAISE(256,COMPID,6,APD,12,APAY,1,OK4RAISE) ;
    END

TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME DEPARTMENT
IF OK4RAISE EQ '1'
END
```

1. The first input field is 256 bytes long. Data is passed in field COMPID. COMPID is the concatenation of several character fields passed as the first parameter. Each of the other parameters is a single argument.
2. The second input field is six bytes long. Data is passed in field APD. The pay date is converted to an alphanumeric field.
3. The third input field is 12 bytes long. Data is passed in field APAY. The monthly gross pay is converted to an alphanumeric field.
4. The return field is up to one byte long and is named OK4RAISE.

The output is:

```
EMP_ID      FIRST_NAME          LAST_NAME       DEPARTMENT
------      ----------          ---------       ----------
071382660   ALFRED              STEVENS         PRODUCTION
```

The FUSREXX macro is displayed below. Commas separate FUSREXX parameters. The ARG command specifies multiple variable names before the first comma and, therefore, separates the first FUSREXX parameter into separate REXX variables, using blanks as delimiters between the variables.

```
/* OK4RAISE routine. Parse separate tokens in 1st parm, then more parms*/
Arg fname lname dept empid,  pay_date,  gross_pay, .
If dept = 'PRODUCTION' & pay_date < '820000'
Then retvalue = '1'
Else retvalue = '0'
Return retvalue
```

FUSREXX routines *should* use the REXX RETURN function to return data to FOCUS. REXX EXIT is acceptable, but is generally used to end an EXEC, not a FUNCTION.

```
Correct
/* Some FUSREXX function */
Arg input
some rexx process ...
Return data_to_Focus
```

```
Not as Clear
/* Another FUSREXX function */
Arg input
some rexx process ...
Exit 0
```

Example Returning an Integer Value

It is possible for REXX to return a value that is *not* character format. The following example shows how REXX returns an integer value. This example also shows how the format of the integer field is used as the last field in the return argument. It passes two input fields and one return field. The FUSREXX routine NUMDAYS returns the number of days between hire date and date of increase. Note that the return value for an integer is *always* four bytes long.

```
    DEFINE FILE EMPLOYEE
1.  AHDT/A6 = EDIT(HIRE_DATE) ;
2.  ADI/A6 = EDIT(DAT_INC) ;
3.  BETWEEN/I6 = NUMDAYS(6,AHDT,6,ADI,4,'I6') ;
    END

TABLE FILE EMPLOYEE
PRINT LAST_NAME HIRE_DATE DAT_INC BETWEEN
IF BETWEEN NE 0
END
```

1. The first input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The second input field is six bytes long. Data is passed in field ADI. The date of increase is converted to an alphanumeric field.
3. The return field is up to six bytes long and is named BETWEEN.

The output is:

```
LAST_NAME       HIRE_DATE   DAT_INC  BETWEEN
---------       ---------   -------  -------
STEVENS         80/06/02    82/01/01     578
STEVENS         80/06/02    81/01/01     213
SMITH           81/07/01    82/01/01     184
JONES           82/05/01    82/06/01      31
SMITH           82/01/04    82/05/14     130
IRVING          82/01/04    82/05/14     130
MCCOY           81/07/01    82/01/01     184
MCKNIGHT        82/02/02    82/05/14     101
GREENSPAN       82/04/01    82/06/11      71
CROSS           81/11/02    82/04/09     158
```

The FUSREXX macro is displayed below. The return value is converted from REXX character to HEX and formatted to be four bytes long.

```
/* NUMDAYS. Return number of days between 2 dates in yymmdd format */
/* The value returned will be in hex format
*/
Arg first,second .
base1 = Date('B',Translate('34/56/12',first,'123456'),'U')
base2 = Date('B',Translate('34/56/12',second,'123456'),'U')
Return D2C(base2 - base1,4)
```

Example Returning a Date Field From a FUSREXX Macro

FOCUS smart date fields contain the integer number of days since the base date 12/31/1900. REXX has a date function that can accept and return several types of date formats, including one called Base format (‘B’) that contains the number of days since the REXX base date 01/01/0001 (Jan. 1 of the Year 1).

Because input arguments must be alphanumeric, you cannot pass a smart date field to a REXX subroutine. Therefore, you can either:

- Pass the REXX routine an alphanumeric field with date display options and have it return a smart date value, if you account for the number of days difference between the FOCUS base date and the REXX base date and convert the result to integer.
- Pass the REXX routine a smart date value converted to alphanumeric format. With this technique, you must account for the difference in base dates for both the input and output.

The following example uses the technique of passing the subroutine an alphanumeric field with date display options. The FUSREXX macro called DATEREX1 takes two input arguments: an alphanumeric date in A8YYMD format and a number of days in character format. It returns a smart date in YYMD format that represents the input date plus the number of days. The FOCUS format A8YYMD corresponds to the REXX Standard format (‘S’).

The number 693959 represents the number of days difference between the FOCUS base date and the REXX base date:

```
/* REXX DATEREX1 routine. Add indate (format A8YYMD) to days */
Arg indate, days .
Return D2C(Date('B',indate,'S')+ days - 693959, 4)
```

The following request uses the DATEREX1 macro to calculate the date that is 365 days from the hire date of each employee. The input arguments are the hire date and the number of days to add. Because HIRE_DATE is in I6YMD format, it must be converted to A8YYMD before being passed to the macro:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE
AND COMPUTE
  ADATE/YYMD =  HIRE_DATE; NOPRINT
AND COMPUTE
  INDATE/A8YYMD= ADATE; NOPRINT
AND COMPUTE
  NEXT_DATE/YYMD = DATEREX1(8,INDATE,3,'365',4,NEXT_DATE);
BY LAST_NAME NOPRINT
END
```

The output is:

```
LAST_NAME        FIRST_NAME  HIRE_DATE  NEXT_DATE
---------        ----------  ---------  ---------
BANNING          JOHN         82/08/01  1983/08/01
BLACKWOOD        ROSEMARIE    82/04/01  1983/04/01
CROSS            BARBARA      81/11/02  1982/11/02
GREENSPAN        MARY         82/04/01  1983/04/01
IRVING           JOAN         82/01/04  1983/01/04
JONES            DIANE        82/05/01  1983/05/01
MCCOY            JOHN         81/07/01  1982/07/01
MCKNIGHT         ROGER        82/02/02  1983/02/02
ROMANS           ANTHONY      82/07/01  1983/07/01
SMITH            MARY         81/07/01  1982/07/01
SMITH            RICHARD      82/01/04  1983/01/04
STEVENS          ALFRED       80/06/02  1981/06/02
```

The following example uses the technique of passing the subroutine a smart date converted to alphanumeric format. The FUSREXX macro called DATEREX2 takes two input arguments: an alphanumeric number of days that represents a smart date, and a number of days to add. It returns a smart date in YYMD format that represents the input date plus the number of days. Both the input date and output date are in REXX base date (‘B’) format.

The number 693959 represents the number of days difference between the FOCUS base date and the REXX base date:

```
/* REXX DATEREX2 routine. Add indate (original format YYMD) to days */
Arg indate, days .
Return D2C(Date('B',indate+693959,'B') + days - 693959, 4)
```

The following request uses the DATEREX2 macro to calculate the date that is 365 days from the hire date of each employee. The input arguments are the hire date and the number of days to add. Because HIRE_DATE is in I6YMD format, it must be converted to an alphanumeric number of days before being passed to the macro:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE
AND COMPUTE
  ADATE/YYMD =  HIRE_DATE; NOPRINT
AND COMPUTE
  INDATE/A8 = EDIT(ADATE); NOPRINT
AND COMPUTE
  NEXT_DATE/YYMD = DATEREX2(8,INDATE,3,'365',4,NEXT_DATE);
BY LAST_NAME NOPRINT
END
```

The report output is the same as that produced by the DATEREX1 macro.

Compiling FUSREXX Macros in CMS

The SUM2 FUSREXX macro takes two amounts as input and returns the sum in integer format:

```
/* SUM2 routine. Add amount1 to amount2 and return as integer */
Arg amt1, amt2 .
Return D2C(amt1 + amt2,4)
```

To compile and compress this FUSREXX macro in CMS, issue the following command. Note that the file identifier must be in upper case:

```
rexxcomp SUM2 FUSREXX A (condense
```

A FILELIST of SUM2 * A lists the following files:

```
SUM2     CFUSREXX A1 F       1024          2          1  1/31/00 12:07:19
SUM2     LISTING  A1 V        121         42          1  1/31/00 12:07:19
SUM2     FUSREXX  A1 F         80          3          1  1/31/00 12:04:19
```

The file SUM2 FUSREXX is the original source file. The file SUM2 CFUSREXX is the compiled version. To call the compiled version in a FOCUS request, you must rename it to have the file type FUSREXX. The file SUM2 LISTING details the results of the compilation.

To use the compiled version in a FOCUS request, issue the following commands. The EXECLOAD command, which loads the routine into memory and improves performance, is optional:

```
rename sum2 fusrexx a ssum2 fusrexx a
rename sum2 cfusrexx a sum2 fusrexx a
execload sum2 fusrexx a
```

Then, in FOCUS, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT CSAL AND COMPUTE
ASAL/A12 = EDIT(CSAL);
AMOUNT/A4 = '1000';
TOTSAL/I6 = SUM2(12, ASAL, 4, AMOUNT, 4, TOTSAL);
END
```

The output is:

```
  CURR_SAL  ASAL          AMOUNT  TOTSAL
  --------  ----          ------  ------
$11,000.00  000000011000  1000     12000
$13,200.00  000000013200  1000     14200
$18,480.00  000000018480  1000     19480
 $9,500.00  000000009500  1000     10500
$29,700.00  000000029700  1000     30700
$26,862.00  000000026862  1000     27862
$21,120.00  000000021120  1000     22120
$18,480.00  000000018480  1000     19480
$21,780.00  000000021780  1000     22780
$16,100.00  000000016100  1000     17100
 $9,000.00  000000009000  1000     10000
$27,062.00  000000027062  1000     28062
```

NF731: Reporting From Independent Paths

When you report from a multi-path data source, a parent segment may have children down some paths but not others. The new 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. In prior releases, WHERE or IF tests on separate paths were considered independently, as if an OR operator connected them. Therefore, a parent instance was included in the report if at least one of the paths passed its screening test. However the following warning message was produced in those cases:

(FOC144) **WARNING. TESTING IN INDEPENDENT SETS OF DATA:**

The request contains a test on one path, and retrieves data from another independent path. Records on the independent path will be retrieved regardless of whether the condition is satisfied on the tested path. Setting MULTIPATH = COMPOUND will alter this behavior.

This behavior is consistent with the SIMPLE setting for the MULTIPATH parameter.

The COMPOUND setting for MULTIPATH eliminates the (FOC144) message when testing on independent paths. It also treats screening conditions on separate paths as if they were connected by an AND operator. That is, *all* paths must pass their screening tests in order for the parent to be included in the report output.

Retrieving Data From Multiple Paths

The examples for this feature use the following segments from the EMPLOYEE data source:

Consider the following request that retrieves data from both paths with MULTIPATH = SIMPLE:

```
SET ALL = OFF
SET MULTIPATH = SIMPLE
TABLE FILE EMPLOYEE
PRINT GROSS IN 18 DATE_ATTEND COURSE_NAME
BY LAST_NAME BY FIRST_NAME IN 12
WHERE PAY_DATE EQ 820831
WHERE COURSE_CODE EQ '103'
END
```

The following warning message is generated:

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

John Banning has taken no courses but he is included in the report output because he has an instance on one of the two paths:

```
LAST_NAME  FIRST_NAME      GROSS DATE_ATTEND COURSE_NAME
---------  ----------      ----- ----------- -----------
BANNING    JOHN        $2,475.00           . .
BLACKWOOD  ROSEMA      $1,815.00           . .
CROSS      BARBAR      $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
MCCOY      JOHN        $1,540.00           . .
MCKNIGHT   ROGER       $1,342.00           . .
ROMANS     ANTHON      $1,760.00           . .
SMITH      MARY        $1,100.00    81/11/16 BASIC REPORT PREP FOR PROG
           RICHAR        $791.67           . .
STEVENS    ALFRED        $916.67           . .
```

If you run the same request with MULTIPATH = COMPOUND, John Banning is omitted from the report output because he has no instances for COURSE_NAME:

```
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
```

The (FOC144) warning message is not generated.

Syntax How to Control Multi-path Processing

Issue the SET MULTIPATH command in one of the following ways:

- From the command level or in a stored procedure:

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

- In a report request:

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

where:

`SIMPLE`

Includes a parent segment in the report output if:

- It has at least one child that passes its screening conditions.
- It lacks any referenced child on a path, but the child is optional (see *Determining if a Segment Is Required*).

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

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

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

The request contains a test on one path, and retrieves data from another independent path. Records on the independent path will be retrieved regardless of whether the condition is satisfied on the tested path. Setting MULTIPATH = COMPOUND will alter this behavior.

COMPOUND

Includes a parent in the report output if it has *all* of its required children (see *Determining if a Segment Is Required*). The COMPOUND setting does not generate the (FOC144) warning message. COMPOUND is the default value for EDA and WebFOCUS.

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

MULTIPATH and SET ALL Combinations

The ALL parameter also 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

`SET ALL = OFF`
`PRINT EMP_ID PAY_DATE`
`  DATE_ATTEND`
	Shows employees who have *either* SALINFO data *or* ATTNDSEG data.	Shows employees who have *both* SALINFO *and* ATTNDSEG data.

`SET ALL = ON`
`PRINT EMP_ID PAY_DATE`
`  DATE_ATTEND`
	Shows employees who have SALINFO data *or* ATTNDSEG data *or* no child data at all.	Same as SIMPLE.

`SET ALL = OFF`
`PRINT EMP_ID PAY_DATE`
`  DATE_ATTEND`
`WHERE PAY_DATE EQ`
`980115`
	Shows employees who have *either* SALINFO data for 980115 *or* any ATTNDSEG data.
	Produces (FOC144) message.
		Shows employees who have *both* SALINFO data for 980115 *and* ATTNDSEG data.

`SET ALL = ON`
`PRINT EMP_ID PAY_DATE`
`  DATE_ATTEND`
`WHERE PAY_DATE EQ`
`980115`
	Shows employees who have *either* SALINFO data for 980115 *or* any ATTNDSEG data.
	Produces (FOC144) message.
		Shows employees who have SALINFO data for 980115. Any DATE_ATTEND data is also shown.

Request	MULTIPATH=SIMPLE	MULTIPATH=COMPOUND

`SET ALL = OFF`
`PRINT ALL.EMP_ID`
`  DATE_ATTEND`
`WHERE PAY_DATE EQ`
`980115`
	Shows employees who have *either* SALINFO data for 980115 *or* any ATTNDSEG data.
	Produces (FOC144) message.
		Shows employees who have SALINFO data for 980115. Any DATE_ATTEND data is also shown.

`SET ALL = ON or OFF`
`PRINT EMP_ID PAY_DATE`
`  DATE_ATTEND`
`WHERE PAY_DATE EQ`
`980115`
`  AND COURSE_CODE EQ`
`'103'`
	Shows employees who have *either* SALINFO data for 980115 *or* COURSE 103. **Note:** SIMPLE treats the AND in the WHERE clause as an OR.
	Produces (FOC144) message.
		Shows employees who have *both* SALINFO data for 980115 *and* COURSE 103.

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

Determining if a Segment Is Required

FOCUS determines if a segment is required or optional using the following rules:

- 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 IF or WHERE 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.

Reference Environments That Support MULTIPATH = COMPOUND

- The MULTIPATH setting works with all types of data sources and in all reporting environments (TABLE, TABLEF, AUTOTABLEF, MATCH, MORE, GRAPH, requests with multiple display commands). It works with alternate views, indexed views, filters, DBA, and joined structures.
- A unique segment is considered a part of its parent segment and, therefore, does not invoke independent path processing.

Reference Requirements and 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 the following message is returned:

  (FOC36263) Insufficient memory for independent path cache.

  There is not enough memory to build a cache for independent paths. Processing will proceed as if MULTIPATH were SIMPLE.

  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 the following message is returned:

  (FOC36264) A segment's active fields are larger than the cache limit.

  One segment has more than 4000 bytes of active data. Processing will proceed as if MULTIPATH were SIMPLE.

- 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.

NF749: HOLD FORMAT INTERNAL

FOCUS HOLD files pad binary and packed 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 FOCUS extract files in an exact manner, you can save the fields in the HOLD file without any padding.

Syntax How to Save a File Without Padding

Issue a report request specifying format overrides for the integer and packed fields that should not be padded and include the following:

```
ON TABLE HOLD AS name FORMAT INTERNAL
```

Syntax How to Ensure Accurate Display of Your Request

```
SET HOLDLIST = PRINTONLY
```

Setting HOLDLIST to PRINTONLY causes your report request to propagate the HOLD file with only the fields that would display in the report output as you specified. If you do not issue this setting, an extra field containing the padded field length is included in the HOLD file.

Reference How to Use Format Overrides

1. Integer fields (I) of one, two, three, or four bytes produce four byte integers.
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. Alpha fields are also not affected by HOLD FORMAT INTERNAL.

3. For packed decimal fields (Px.y), x is the total number of digits and y is the number of digits to the right of the decimal point. The number of bytes is derived by dividing x by 2 and adding 1.

```
bytes = INT (x/2) + 1
```

where:

INT (x/2) is the greatest integer after dividing by 2.

4. Alphanumeric fields automatically inherit their length from their source Master File and are not padded to a full word boundary.

Example Creating a HOLD File Without HOLD FORMAT INTERNAL

```
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
SEGNAME=DJG     ,SEGTYPE=S0
FIELDNAME   =CAR                ,E01         ,A16       ,A16       ,$
FIELDNAME   =COUNTRY            ,E02         ,A10       ,A12       ,$
FIELDNAME   =RETAIL_COST        ,E03         ,D7        ,D08       ,$
FIELDNAME   =DEALER_COST        ,E04         ,D7        ,D08       ,$
FIELDNAME   =SEATS              ,E05         ,I3        ,I04       ,$
```

The values of ACTUAL for RETAIL_COST, DEALER_COST, and SEATS are all padded to a full word in binary.

Example Creating a HOLD File With HOLD FORMAT INTERNAL

In this example, DEALER_COST and RETAIL_COST are defined in the Master File as D fields, but the request overrides RETAIL_COST as an I2 field and DEALER_COST as a P3 field.

```
SET HOLDLIST=PRINTONLY
TABLE FILE CAR
PRINT CAR COUNTRY RETAIL_COST/I2 DEALER_COST/P3 SEATS/I1
ON TABLE HOLD AS HINT3 FORMAT INTERNAL
END
```

This creates the following Master File:

```
FILE=HINT3          ,SUFFIX=FIX
SEGNAME=HINT3   ,SEGTYPE=S0
FIELDNAME   =CAR                ,E01         ,A16       ,A16       ,$
FIELDNAME   =COUNTRY            ,E02         ,A10       ,A10       ,$
FIELDNAME   =RETAIL_COST        ,E03         ,I6        ,I02       ,$
FIELDNAME   =DEALER_COST        ,E04         ,P4        ,P02       ,$
FIELDNAME   =SEATS              ,E05         ,I4        ,I01       ,$
```

The values of ACTUAL for the overridden fields are I2 and P2. DEALER_COST has an ACTUAL of P2 because the format override, P3, means 3 display digits that can be stored in 2 actual digits.

If a format override is not large enough to contain the data values, the values are truncated. Truncation may cause the data in the HOLD file to be incorrect in the case of an integer. For packed data and integers, truncation occurs for the high order digits so the remaining low order digits will resemble the digits from the correct values.

For example, consider this next request:

```
SET HOLDLIST=PRINTONLY
TABLE FILE CAR
PRINT CAR COUNTRY RETAIL_COST/I1 DEALER_COST/P3 SEATS/I1
ON TABLE HOLD AS HINT4 FORMAT INTERNAL
END
```

RETAIL_COST and SEATS are overridden with format I1 and DEALER_COST is overridden with format P3. These formats for RETAIL_COST and DEALER_COST are not large enough to contain the data values for these fields.

The following Master File is produced:

```
FILE=HINT4          ,SUFFIX=FIX
SEGNAME=HINT4    ,SEGTYPE=S0
FIELDNAME   =CAR                ,E01         ,A16       ,A16       ,$
FIELDNAME   =COUNTRY            ,E02         ,A10       ,A10       ,$
FIELDNAME   =RETAIL_COST        ,E03         ,I4        ,I01       ,$
FIELDNAME   =DEALER_COST        ,E04         ,P4        ,P02       ,$
FIELDNAME   =SEATS              ,E05         ,I4        ,I01       ,$
```

The following is a sampling of the output when reporting from this HOLD file:

CAR	COUNTRY	RETAIL_COST	DEALER_COST	SEATS
JAGUAR	ENGLAND	174	427	2
JAGUAR	ENGLAND	179	194	5
JENSEN	ENGLAND	186	940	4
TRIUMPH	ENGLAND	236	296	2
DATSUN	JAPAN	67	626	4
TOYOTA	JAPAN	11	886	4
ALFA ROMEO	ITALY	164	660	2
ALFA ROMEO	ITALY	164	660	2
ALFA ROMEO	ITALY	37	915	4

The values displayed for RETAIL_COST and DEALER_COST do not represent the actual values in the original data source. This is due to truncation of the binary integer representation of the data values for RETAIL_COST, and truncation of the high order digits for DEALER_COST.

For example, a RETAIL_COST of 8878 is 10001010101110 in binary. To fit this into an I1 field as stated in the prior TABLE request, the value is 10101110. So the high order 100010 is truncated. Now, the decimal value of 100010 is 174, which is why the first record displayed shows a RETAIL_COST of 174.

For a DEALER_COST of 7427 to fit in a P2 ACTUAL, the high order digit, 7, is truncated, leaving the low order digits, 427. This is why the first record displayed shows a DEALER_COST of 427.

NF750: DATASET in a Master File

You can add the DATASET attribute to the Master File to specify a physical location for the data source to be allocated. In addition, the DATASET attribute permits you to bypass the FOCUS search mechanism for default data source location. DATASET eliminates the need to allocate data sources using JCL, FILEDEF, DYNAM, and USE commands.

User allocation and system specific behavior is as follows:

Platform	User allocation command
CMS	FILEDEF
TSO	DYNAM ALLOC or TSO ALLOC

Note: The MODIFY FIND function does not work with the DATASET attribute. To use FIND with a data source, you must manually allocate the data source.

DATASET Behavior in FOCUS Data Sources

The DATASET attribute can be used only on the file level of the Master File. If the Master File's name is present in the USE list, or the user explicitly allocated the Master File, a warning is issued and the DATASET attribute is ignored.

If DATASET is used in a Master File whose data source is managed by the FOCUS Database Server, the DATASET attribute is ignored on the server side because the FOCUS Database Server does not read Master Files for servicing table requests.

The DATASET attribute in the Master File has the lowest priority:

- A user's explicit allocation overrides DATASET attributes.
- The USE command for FOCUS data sources overrides DATASET attributes and explicit allocations.

An alternative to the DATASET attribute for allocating FOCUS data sources is an Access File. For detailed information, see NF777: Partitioned FOCUS Data Sources. DATASET and ACCESSFILE are mutually exclusive attributes in the Master File; that is, you can use at most one of them, not both.

Note: If a DATASET allocation is in effect, a CHECK FILE command must be issued in order to override it by an explicit allocation command. The CHECK FILE command will de-allocate the allocation created by DATASET.

Syntax How to Use the DATASET Attribute

```
{DATASET|DATA}='filename [ON sinkname]'
```

where:

filename

Is the platform-dependent physical name of the data source.

sinkname

Indicates that the data source is located on the FOCUS Database Server. This attribute is valid for FOCUS data sources.

In MVS, the syntax is:

```
{DATASET|DATA}='qualifier.qualifier ...'
```

or

```
{DATASET|DATA}='ddname ON sinkname'
```

In CMS, the syntax is:

```
{DATASET|DATA}='filename filetype filemode [ON sinkname]'
```

Example Allocating a FOCUS Data Source Using the DATASET Attribute

The following example illustrates how to allocate a FOCUS data source using the DATASET attribute:

For MVS,

```
FILENAME=CAR,SUFFIX=FOC
DATASET='USER1.CAR.FOCUS'
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
.
.
.
```

For CMS,

```
FILENAME=CAR,SUFFIX=FOC
DATASET='CAR FOCUS A'
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
.
.
.
```

Example Allocating a Data Source For the FOCUS Database Server

The following example illustrates how to allocate a FOCUS data source with the DATASET attribute using ON *sink*:

For MVS,

```
FILENAME=CAR,SUFFIX=FOC
DATASET='CAR ON SINK1'
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
.
.
.
```

Note: The ddname CAR is allocated by the FOCUS Database Server JCL.

For CMS,

```
FILENAME=CAR,SUFFIX=FOC
DATASET='CAR FOCUS A ON SINK1'
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
.
.
.
```

DATASET Behavior in Fixed-Format Sequential Data Sources

The DATASET attribute must appear on the file level of the Master File and cannot contain ON *sink*. If the DATASET attribute contains ON *sink*, an error message is issued and the operation is terminated.

When FOCUS detects the DATASET attribute, FOCUS checks for an explicit allocation of data for this Master File. If an explicit allocation exists, a warning message is issued informing the user that the DATASET value has been overridden and the DATASET attribute is ignored. If this Master File name is not allocated, an internal command is issued to perform the allocation. This allocation is stored temporarily and is released when a new Master File is used or when the FOCUS session terminates.

Syntax How to Use the DATASET Attribute With Fixed-Format Data Sources

```
{DATASET|DATA}='filename'
```

where:

filename

Is the platform-dependent physical name of the data source.

The DATASET attribute in the Master File has the lowest priority:

- A user's explicit allocation overrides DATASET attributes.

Note: If a DATASET allocation is in effect, a CHECK FILE command must be issued in order to override it by an explicit allocation command. The CHECK FILE command will de-allocate the allocation created by DATASET.

Example Allocating a Fixed-Format Data Source Using the DATASET Attribute

The following example illustrates how to allocate a fixed-format data source using the DATASET attribute:

1. `FILE=XX,  SUFFIX=FIX,  DATASET='SEQFILE1 DATA A'`

   .
   .
   .

2. `FILE=XX,  SUFFIX=FIX,  DATASET='USER1.SEQFILE1'`

   .
   .
   .

DATASET Behavior in VSAM Data Sources

The DATASET attribute must appear on the file level of the Master File and cannot contain ON *sink*. If the DATASET attribute contains ON *sink*, an error message is issued and the operation is terminated.

When FOCUS detects the DATASET attribute, FOCUS checks for an explicit allocation of data for this Master File. If an explicit allocation is found, a warning message is issued informing the user that the DATASET value has been overridden and the DATASET attribute is ignored. If this Master File name is not allocated, an internal command is issued to perform the allocation. This allocation is stored temporarily and is released when a new Master File is used or when the FOCUS session terminates.

The DATASET attribute may also appear on the field level of the Master File to specify where to find an alternate index. Because of VSAM naming conventions (truncated to 8 characters), the name of the field alias will be used as the ddname. If a user allocation is found for the Master File or alternate index ddname, the DATASET attribute is ignored and a warning message issued.

Note: There is no limit on how many alternate indices you may have. It is also acceptable for some alternate indices to have the DATASET attribute and others not. However, if a file level DATASET attribute is missing, the field level DATASET will be ignored.

Syntax How to Use the DATASET Attribute With VSAM Data Sources

`{DATASET|DATA}='filename'`

where:

filename

Is the platform-dependent physical name of the data source or alternate index.

The DATASET attribute in the Master File has the lowest priority:

- A user's explicit allocation overrides DATASET attributes.

Note: If a DATASET allocation is in effect, a CHECK FILE command must be issued in order to override it by an explicit allocation command. The CHECK FILE command will de-allocate the allocation created by DATASET.

Example Allocating a VSAM Data Source Using the DATASET Attribute

The following example illustrates how to allocate a VSAM data source on the file level and for an alternate index:

```
FILE=EXERVSM1,  SUFFIX=VSAM, DATASET='VSAM1.CLUSTER1',$
SEGNAME=ROOT , SEGTYPE=S0,$
 GROUP=KEY1  , ALIAS=KEY , FORMAT=A4, ACTUAL=A4 ,$
  FIELD=FLD1 , ALIAS=F1  , FORMAT=A4, ACTUAL=A4 ,$
  FIELD=FLD2 , ALIAS=F2  , FORMAT=A4, ACTUAL=A4 ,$
  FIELD=FLD3 , ALIAS=DD1 , FORMAT=A4, ACTUAL=A4 , FIELDTYPE = I ,
        DATASET='VSAM1.INDEX1' ,$
  FIELD=FLD4 , ALIAS=F4  , FORMAT=A4, ACTUAL=A4 ,$
  FIELD=FLD5 , ALIAS=F5  , FORMAT=A4, ACTUAL=A4 ,$
  FIELD=FLD6 , ALIAS=F6  , FORMAT=A4, ACTUAL=A4 ,$
  FIELD=FLD7 , ALIAS=F7  , FORMAT=A4, ACTUAL=A4 ,$
```

Reference Error Messages

(FOC1920) ERROR ALLOCATING FILE %1 AS %2

System could not complete an internal allocation command. Check file name, allocation, and attributes.

(FOC1921) ERROR PARSING DATASET IN MASTER %1: %2

Error occurred while analyzing value of DATASET. Check the Master File for spelling errors.

(FOC1922) FOCUS DATASET INTERNAL ERROR: %1

A serious problem occurred processing the FOCUS DATASET.

(FOC1925) DATASET ERROR: PHYSICAL NAME SPECIFIED WITH 'ON' CLAUSE

Use a ddanme instead of the physical file name in DATASET when ON *sinkname* clause is present.

Reference Warning Messages

(FOC1923) WARNING: USE COMMAND OVERRIDES DATASET VALUE IN %1

DATASET value in Master File has been overridden by USE command.

(FOC1924) WARNING: ALLOCATION OF %1 OVERRIDES DATASET VALUE IN %1

DATASET value in Master File has been overridden by user's allocation.

NF751: Date-Time Data Type

Topics:

- Describing Date-Time Values
- Setting Date-Time Options
- Manipulating Date-Time Values

The new date-time data type supports both the date and time, similar to the timestamp data types available in many relational data sources.

Date-time fields are stored in eight or ten bytes, four digits for date and either four or six digits for time, depending on whether the format specifies microseconds.

New subroutines are provided for manipulating date-time fields.

Describing Date-Time Values

In a Master File, the USAGE format for a date-time field describes which components to display and various options for displaying them. In Master Files for non-FOCUS data sources, date-time fields must also have an ACTUAL format that indicates how the date-time value is stored in the non-FOCUS data source.

The MISSING attribute for date-time fields can be ON or OFF. If it is OFF, and the date-time field has no value, it defaults to blank.

This section discusses:

- USAGE formats for displaying date-time field values.

- Alphanumeric formats for date-time values entered by a user at a terminal, read from a transaction file, or embedded in an expression.
- ACTUAL formats for date-time fields.

Date-Time Display Formats

The USAGE (or FORMAT) attribute determines how date-time field values are displayed in report output and forms, and how they behave in expressions and functions; for FOCUS data sources, it also determines how they are stored. A new format type, H, describes date-time fields. The USAGE attribute for a date-time field contains the H format code and can identify either the length of the field or the relevant date-time display options.

The USAGE attribute can be one of the following

```
USAGE = Hnn
USAGE = Htimefmt1
USAGE = Hdatefmt [separator] [timefmt2]
```

where:

`Hnn`

Is the USAGE value for a numeric date-time value without date-time display options. This format is appropriate for use in alphanumeric HOLD files or transaction files.

nn is the field length, from 1 to 20, including up to eight characters for displaying the date and up to nine or 12 characters for the time. For lengths less than 20, the date is truncated on the right.

An eight-character date includes four digits for the year, two digits for the month, and two digits for the day of the month, YYYYMMDD.

A nine-character time includes two digits for the hour, two digits for the minute, two digits for the second, and three digits for the millisecond, HHMMSSsss. The millisecond component represents the decimal portion of the second to three places.

A twelve-character time includes two digits for the hour, two digits for the minute, two digits for the second, three digits for the millisecond, and three digits for the microsecond, HHMMSSsssmmm. The millisecond component represents the decimal portion of the second value to three places. The microsecond component represents three additional decimal places beyond the millisecond value.

With this format, there are no spaces between the date and time components, no decimal points, and no spaces or separator characters within either component. The time must be entered using the 24-hour system. For example, the value `19991231225725333444` represents `1999/12/31 10:57:25.333444PM`

Htimefmt1

Is the USAGE format for displaying time only. Hour, minute, and second components are always displayed separated by colons (:), with no intervening blanks.

Unless you specify one of the AM/PM time display options, the time component is displayed using the 24-hour system.

When the format includes more than one time display option:

- The options must appear in the order hour, minute, second, millisecond, microsecond.
- The first option must be either hour, minute, or second.
- No intermediate component can be skipped. That is, if hour is specified the next option must be minute, it cannot be second.

The following table lists the valid time display options for a time-only USAGE attribute. Assume the time value is 2:05:27.123456 a.m.

Option	Meaning	Effect
H	hour (two digits) If the format includes the option `a` or `A`, the hour value is from 01 to 12. Otherwise, the hour value is from 00 to 23, with 00 representing midnight.	Prints a two-digit hour. For example: `USAGE = HH` prints `02`
h	hour with zero suppression If the format includes the option `a` or `A`, the hour value is from 1 to 12. Otherwise, the hour is from 0 to 23.	Displays the hour with zero suppression. For example: `USAGE = Hh` prints `2`
I	minute (two digits) The minute value is from 00 to 59.	Prints the two-digit minute. For example: `USAGE = HHI` prints `02:05`

Option	Meaning	Effect
i	minute with zero suppression The minute value is from 0 to 59.	Prints the minute with zero suppression. Cannot be used together with an hour format (H or h). For example: `USAGE = Hi` prints `5`
S	Second (two digits) 00 to 59	Prints the two-digit second. For example: `USAGE = HHIS` prints `02:05:27`
s	millisecond (three digits — after the decimal point in the second) 000 to 999	Prints the second to three decimal places. For example: `USAGE = HHISs` prints `02:05:27.123`
m	microsecond (three additional digits after milliseconds) 000 through 999	Prints the second to six decimal places. For example: `USAGE = HSsm` prints `27.123456`
A	12-hour time display with AM or PM in upper case	Prints hours from 01 to 12 followed by AM or PM. For example: `USAGE = HHISA` prints `02:05:27AM`

Option	Meaning	Effect
a	12-hour time display with am or pm in lower case	Prints hours from 01 to 12 followed by am or pm. For example: `USAGE = HHISa` prints `02:05:27am`

`Hdatefmt`

Is the USAGE format for displaying the date portion of the date-time field.

The date components can be in any of the following combinations and order:

- Year first combinations: Y, YY, YM, YYM, YMD, YYMD
- Month-first combinations: M, MD, MY, MYY, MDY, MDYY
- Day-first combinations: D, DM, DMY, DMYY

The date format can include the following display options as long as they conform to the allowed combinations. In the following table, assume the date is February 5, 1999.

Option	Meaning	Example
Y	2-digit year	`99`
YY	4-digit year	`1999`
M	2-digit month (01 - 12)	`02`
MT	Full month name	`February`

Option	Meaning	Example
Mt	Short month name	`Feb`
D	2-digit day	`05`
d	zero-suppressed day	`5`
k	For formats in which month or day is followed by year and month is translated to a short or full name, separates the year from the day with a comma and blank. Otherwise the separator is a blank.	`USAGE = HMtDkYY` prints `Feb 05, 1999`

separator

Is a separator between the date components. The default separator is a slash (/). Other valid separators are: period (.), hyphen (-), blank (B), or none (N). With translated months, these separators can only be specified when the k option is not used.

timefmt2

Is the format for a time that follows a date. Time is separated from the date by a blank; time components are separated from each other by colons. Unlike the format for time alone, a time format that follows a date format consists of at most two characters: a single character to represent all of the time components to be displayed and, optionally, one character for an AM/PM option. The following table lists the valid options. Assume the date is February 5, 1999 and the time is 02:05:25.444555 a.m.

Option	Meaning	Example
H	Prints hour	`USAGE = HYYMDH` prints `1999/02/05 02`
I	Prints hour:minute	`USAGE = HYYMDI` prints `1999/02/05 02:05`
S	Prints hour:minute:second	`USAGE = HYYMDS` prints `1999/02/05 02:05:25`
s	Prints hour:minute:second.millisecond	`USAGE = HYYMDs` prints `1999/02/05 02:05:25.444`
m	Prints hours:minutes:seconds.microseconds	`USAGE = HYYMDm` prints `1999/02/05 02:05:25.444555`

Option	Meaning	Example
A	Prints AM or PM. Uses the 12-hour system and causes the hour to be printed with zero suppression.	`USAGE = HYYMDSA` prints `1999/02/05 2:05:25AM`
a	Prints am or pm. Uses the 12-hour system and causes the hour to be printed with zero suppression.	`USAGE = HYYMDSa` prints `1999/02/05 2:05:25am`

Note: Unless you specify one of the AM/PM time display options, the time component is displayed using the 24-hour system.

Specifying Date-Time Values

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 clause, an IF clause, a 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

```
date_string   [time_string]
time_string   [date_string]
```

A date-time constant in a COMPUTE, DEFINE, or WHERE expression must have one of the following formats.

```
DT(date_string  [time_string])
DT(time_string  [date_string])
```

A date-time constant in an IF expression has one of the following formats:

```
'date_string  [time_string]'
'time_string  [date_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 seconds can be expressed with a decimal point or be followed by a colon.

- If there is a colon after seconds, the value following it represents milliseconds. There is no way to express microseconds using this notation.
- A decimal point in the seconds value indicates the decimal fraction of a second. Microseconds can be represented using six decimal digits.

date_string

Can have one of the following three formats:

- The **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 Formatted Input Values* is ignored.

  If a numeric-string format longer than eight digits is encountered, it is treated as a combined date-time string in the H*nn* format described in *Describing Date-Time Values*. The following are examples of numeric string date constants:

  ```
  99
  1999
  19990201
  ```

- The **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 Formatted Input Values*. 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
  ```

- The **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.
- In each date format, two-digit years are interpreted using the [F]DEFCENT and [F]YRTHRESH settings.

Example Reading Date-Time Values From a Transaction File

The DTTRANS 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 will receive these values:

```
FILE=DATETIME,     SUFFIX=FOC    ,$
SEGNAME=DATETIME,  SEGTYPE=S0    ,$
FIELD=ID,  ID,     USAGE = I2    ,$
FIELD=DT1, DT1,    USAGE=HYYMDS ,$
```

The following MODIFY procedure loads the transaction values into the FOCUS data source:

```
-* THE FOLLOWING ALLOCATION AND CREATE FILE IS NEEDED FOR THE TRANSACTION
FILE
-* ON MVS
-* DYNAM ALLOC DD DTTRANS DA USER1.DTTRANS SHR REUSE
-* CREATE FILE DATETIME
-* THE FOLLOWING FILEDEF IS NEEDED FOR THE TRANSACTION FILE ON CMS
CMS FILEDEF DTTRANS DISK DTTRANS DATA A
MODIFY FILE DATETIME
  FREEFORM ID DT1
DATA ON DTTRANS
END
```

To see the printed values, issue the following request:

```
TABLE FILE DATETIME
PRINT ID DT1
END
```

The output is:

```
ID  DT1
--  ---
 1  2000/01/01 02:57:25
 2  1999/12/31 14:05:35
```

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
```

ACTUAL Formats for Date-Time Values

ACTUAL formats supported for date-time values are:

- A*nn*, H8, H10, and H12. A*nn* accepts all the date-time string formats described in *How to Specify the Order of Date Components in Formatted Input Values*, as well as the H*nn* USAGE display format described in *Describing Date-Time Values*. ACTUAL=H8, H10, or H12 accepts a date-time field as it occurs in a binary HOLD file or SAVB file. ACTUAL=A*nn* accepts a date-time field as it occurs in an alphanumeric HOLD file or SAVE file.

Example Creating a Binary HOLD File Containing a Date-Time Field

The following request creates a binary HOLD file using the VIDEOTR2 data source:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE
ON TABLE HOLD AS DTHOLD
END
```

The DTHOLD Master File created from this request contains the ACTUAL format H8 for the TRANSDATE date-time field:

```
FILE=DTHOLD          ,SUFFIX=FIX
SEGNAME=DTHOLD   ,SEGTYPE=S0
FIELDNAME    =CUSTID              ,E01          ,A4        ,A04       ,$
FIELDNAME    =TRANSDATE           ,E02          ,HYYMDI    ,H08       ,$
```

Example Creating an Alphanumeric HOLD File Containing a Date-Time Field

The following request creates an alphanumeric HOLD file using the VIDEOTR2 data source:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE
ON TABLE HOLD AS DTALPHA FORMAT ALPHA
END
```

The DTALPHA Master File created from this request contains the ACTUAL format A17 for the TRANSDATE date-time field. The ACTUAL format is A20 if the time format includes microseconds, A17 otherwise (padded with low order zeros if necessary):

```
FILE=DTALPHA         ,SUFFIX=FIX
SEGNAME=DTALPHA ,SEGTYPE=S0
FIELDNAME   =CUSTID              ,E01          ,A4        ,A04       ,$
FIELDNAME   =TRANSDATE           ,E02          ,HYYMDI    ,A17       ,$
```

Setting Date-Time Options

Three parameter settings determine how to interpret a date-time value and manipulate it in requests:

- DATEFORMAT specifies the order of the date components.
- WEEKFIRST assigns a day to be considered the first day of the week in date-time computations involving weeks.
- DTSTRICT controls whether values are error checked (for example, whether the day portion of a date is within the correct number of days for the specified month).

Syntax How to Specify the Order of Date Components in Formatted Input Values

The DATEFORMAT parameter specifies the order of the date components (month/day/year) when date-time values are entered in the formatted string and translated string formats described in *Describing Date-Time Values*. It makes a value's input format independent of the format of the variable to which it is being assigned. The syntax is

```
SET DATEFORMAT = datefmt
```

where:

datefmt

Can be one of the following: MDY, DMY, YMD, or MYD. The U. S. English default format is MDY.

Syntax How to Specify the First Day of the Week

The WEEKFIRST parameter is used in week computations by the HDIFF, HNAME, HPART, and HSETPT functions described in *Date-Time Functions*. For an example, see *Extracting the Week Component With Different WEEKFIRST Settings*. The values from 1 to 7 represent Sunday through Saturday. The syntax is

```
SET WEEKFIRST = number
```

where:

number

Is a number from one to seven, where one represents Sunday and seven represents Saturday. The U. S. English default value is seven (Saturday) meaning that Saturday is the first day of each week, so every Friday-Saturday transition is the start of a new week.

The WEEKFIRST setting does not change the number that corresponds to each day of the week, it just specifies which one is considered the start of the week. The default of Saturday (7) as the first day of the week is consistent with the Microsoft SQL Server convention.

Syntax How to Control Error Checking of Date-Time Values

The DTSTRICT parameter controls how much error checking is done on date-time values when they are input by users, read from an alphanumeric transaction file, displayed, or used in user-written subroutines. The syntax is

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

where:

ON

Invokes strict processing. This means that whenever a date-time value is input by a user, read from a transaction file, displayed, or returned by a subroutine it is checked to make sure that the value represents a valid date and time. For example, a numeric month must be between 1 and 12, and the day must be within the number of days for the specified month. ON is the default value. If you attempt to enter a value that violates this rule, the following message displays:

```
(FOC177) INVALID DATE CONSTANT: dt_constant
```

OFF

Does not invoke strict processing. Any date-time component can have any value within the constraint of the number of decimal digits allowed; for example, the month value can be 00 or 13 or 99, but not 115. Furthermore, the values do not have to be consistent; for example, any month in any year can have 30 or 31 days.

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. All other operations are accomplished through a set of date-time functions.

Comparison and Assignment

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=HYYMDS, 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/HYYMDH = 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
```

Date-Time Functions

The following functions allow you to manipulate date-time values:

Function Name	Description
HCNVRT	Converts date-time values to alphanumeric format for use with operators such as EDIT, CONTAINS, and LIKE.
HINPUT	Converts an alphanumeric string to a date-time value.
HADD	Increments date-time values by a specified number of units.

Function Name	Description
HDIFF	Returns the number of units of a specific date-time component between two date-time values.
HNAME	Extracts specified components of a date-time value and converts them to alphanumeric format.
HPART	Extracts a component of a date-time value in numeric format.
HSETPT	Inserts the numeric value of a specified component in a date-time field.
HMIDNT	Changes the time portion of a date-time field to midnight.
HDATE	Extracts the date components from a date-time field and converts them to a date field.
HDTTM	Converts a date field to a date-time field with the time set to midnight.
HTIME	Extracts all of the time components from a date-time field and converts them to a number of milliseconds or microseconds in numeric format.
HGETC	Returns the current date and time in date-time format.

Note:

- In those arguments that give you a choice of 8 or 10, use 8 for processing values without microseconds, 10 when the field value includes microseconds.
- The last argument is always a USAGE format that indicates the data type returned by the function. The type may be A (alpha), I (integer), D (double precision), DATE (smart date), or H (date-time).

Reference Component Names and Values for Use With Date-Time Functions

The following component names and values are supported as arguments to those date-time functions that require you to specify a component name as an argument:

Component Name	Valid Values
`year`	0001-9999
`quarter`	1-4
`month`	1-12
`day-of-year`	1-366
`day` or `day-of-month`	1-31 (The two names for the component are equivalent.)
`week`	1-53
`weekday`	1-7 (Sunday-Saturday)

Component Name	Valid Values
`hour`	0-23
`minute`	0-59
`second`	0-59
`millisecond`	0-999
`microsecond`	0-999999

Reference Notes Regarding ISO Standard Date-Time Representations

International Standard ISO 8601 describes the standards for numeric representations of date and time. Some of the relevant standards and notes about their implementation follow:

- The international standard date notation is YYYY-MM-DD. In this implementation, you can control the date format used to enter date-time values with the DATEFORMAT parameter. For details, see *How to Specify the Order of Date Components in Formatted Input Values*.
- The international standard for the first day of a week is Monday. You can use the WEEKFIRST parameter to control the day used as the first day of the week by the date-time functions.

- The standard specifies that week 1 of a year is the first week of the year that has a Thursday. Combined with the standard of Monday as day 1, this rule ensures that week 1 has at least four of its days in the specified year.

  The following rules represent an extension to the standard in this implementation:

  - Whatever day you choose for your WEEKFIRST setting, the date-time functions define week 1 as the first week with at least four days in the specified year.
  - With these rules, it is possible for the first few days of January to fall in the week prior to week 1. The international standard considers these dates to be in week 53 of the previous year. However, the date-time functions return zero for the week component when it falls in the week prior to week 1.
- The international standard notation for the time of day is hh:mm:ss using the 24-hour system. However, the date-time data type and date-time functions allow you to use the 12-hour system.

HCNVRT: Converting a Date-Time Field to Alphanumeric Format

Use the following syntax in an expression to convert a date-time field to alphanumeric format for use with operators such as EDIT, CONTAINS, and LIKE.

```
HCNVRT (dtfield, '(Hfmt)', rlength, 'Ann')
```

where:

dtfield

Is the date-time value to convert. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

Hfmt

Is the USAGE format of the date-time field being converted, enclosed in parentheses and single quotation marks.

rlength

Is the length of the alphanumeric field returned. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value. If *rlength* is smaller than the number of characters needed to display the alphanumeric field, an all-blank field is returned.

Ann

Is the USAGE format of the returned alphanumeric value, enclosed in single quotation marks.

Example Converting a Date-Time Field to Alphanumeric Format

The following request converts the TRANSDATE field to alphanumeric format:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME1/A20 = HCNVRT (TRANSDATE,'(H17)', 17, 'A20');
ALPHA_DATE_TIME2/A20 = HCNVRT (TRANSDATE,'(HYYMDS)', 20, 'A20');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         ALPHA_DATE_TIME1      ALPHA_DATE_TIME2
------  ---------         ----------------      ----------------
1118    2000/06/26 05:45  20000626054500000     2000/06/26 05:45:00
1237    2000/02/05 03:30  20000205033000000     2000/02/05 03:30:00
```

HINPUT: Converting an Alphanumeric String to a Date-Time Value

Use the following syntax in an expression to convert an alphanumeric string to a date-time value.

```
HINPUT (inputlength, 'inputstring', {8|10}, 'Hfmt')
```

where:

inputlength

Is the length of the alphanumeric string to convert. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

inputstring

Is the alphanumeric string to convert. You can supply the actual string enclosed in single quotation marks, the name of an alphanumeric field, or an expression that returns an alphanumeric value. The alphanumeric string can consist of any valid date-time input value as described in *Describing Date-Time Values*.

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

H*fmt*

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Converting an Alphanumeric String to a Date-Time Value

The following request converts the TRANSDATE field to alphanumeric format (using the HCNVRT function) and then uses the HINPUT routine to convert the alphanumeric string to a date-time value:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME/A20 = HCNVRT (TRANSDATE,'(H17)', 17, 'A20');
DT_FROM_ALPHA/HYYMDS = HINPUT(14, ALPHA_DATE_TIME, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         ALPHA_DATE_TIME1      ALPHA_DATE_TIME2
------  ---------         ----------------      ----------------
1118    2000/06/26 05:45  20000626054500000     2000/06/26 05:45:00
1237    2000/02/05 03:30  20000205033000000     2000/02/05 03:30:00
```

HADD: Incrementing a Date-Time Field

Use the following syntax in an expression to increment a date-time field by a given number of units, for example, 1 year, 3 months, or -15 seconds.

```
HADD (dtfield, 'component', increment, {8 | 10}, 'Hformat')
```

where:

dtfield

Is the date-time value to increment. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be incremented, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* for a list of supported components.

increment

Is the number of units by which to increment the specified component. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

`8 | 10`

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

`H`*format*

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Incrementing the Month Component of a Date-Time Field

The following request adds two months to the TRANSDATE field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME          ADD_MONTH
------  ---------          ---------
1118    2000/06/26 05:45   2000/08/26 05:45:00
1237    2000/02/05 03:30   2000/04/05 03:30:00
```

If necessary, the day is adjusted to be valid for the resulting month.

HDIFF: Finding the Number of Units Between Two Date-Time Values

Use the following syntax in an expression to find the number of boundaries of a given type crossed in going from date 2 to date 1.

```
HDIFF (dtfield1, dtfield2, 'component', 'Dformat')
```

where:

dtfield1

Is the ending date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

dtfield2

Is the starting date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be used in the calculation, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* for a list of supported components. If the unit is weeks, the WEEKFIRST setting is used in the calculation.

Dformat

Is the USAGE format of the resulting number of units, enclosed in single quotation marks. The format type must be D.

Example Finding the Number of Days Between Two Date-Time Fields

The following request finds the number of days between the ADD_MONTH and TRANSDATE fields:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
DIFF_DAYS/D12.2 =  HDIFF(ADD_MONTH, TRANSDATE, 'DAY', 'D12.2');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         ADD_MONTH                 DIFF_DAYS
------  ---------         ---------                 ---------
1118    2000/06/26 05:45  2000/08/26 05:45:00           61.00
1237    2000/02/05 03:30  2000/04/05 03:30:00           60.00
```

HNAME: Extracting a Date-Time Component in Alphanumeric Format

Use the following syntax in an expression to extract a specified component from a date-time field and return it in alphanumeric format.

```
HNAME (dtfield, component, Aformat)
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be extracted, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* for a list of supported components.

Aformat

Is the alphanumeric USAGE format of the returned component, enclosed in single quotation marks. All other components are converted to strings of digits only. The year is always four digits, and the hour assumes the 24-hour system.

Example Extracting the Day Component in Alphanumeric Format From a Date-Time Field

The following request extracts the day in alphanumeric format from the TRANSDATE field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/A2  = HNAME(TRANSDATE, 'DAY', 'A2');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         DAY_COMPONENT
------  ---------         -------------
1118    2000/06/26 05:45  26
1237    2000/02/05 03:30  05
```

Example Extracting the Week Component With Different WEEKFIRST Settings

The following request extracts the week in alphanumeric format from the TRANSDATE field. Changing the WEEKFIRST setting changes the value of the extracted component:

```
SET WEEKFIRST = 7
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
WEEK_COMPONENT/A10 = HNAME(TRANSDATE, 'WEEK', 'A10');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         WEEK_COMPONENT
------  ---------         --------------
1118    2000/06/26 05:45  26
1237    2000/02/05 03:30  06
```

Running the same request setting WEEKFIRST to 3 produces the following output (see *How to Specify the First Day of the Week*):

```
CUSTID  DATE-TIME          WEEK_COMPONENT
------  ---------          --------------
1118    2000/06/26 05:45   25
1237    2000/02/05 03:30   05
```

HPART: Extracting a Date-Time Component in Numeric Format

Use the following syntax in an expression to extract a specified component from a date-time field and return it in numeric format.

```
HPART (dtfield, 'component', 'Iformat')
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be extracted, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* for a list of supported components.

Iformat

Is the integer USAGE format of the returned component, enclosed in single quotation marks. The year is always four digits, and the hour assumes the 24-hour system.

Example Extracting the Day Component in Numeric Format From a Date-Time Field

The following request extracts the day in integer format from the TRANSDATE field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/I2  = HPART(TRANSDATE, 'DAY', 'I2');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         DAY_COMPONENT
------  ---------         -------------
1118    2000/06/26 05:45             26
1237    2000/02/05 03:30              5
```

HSETPT: Inserting a Component Into a Date-Time Field

Use the following syntax in an expression to insert the numeric value of a specified component into a date-time field.

```
HSETPT (dtfield, 'component', value, {8|10}, 'Hformat')
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be inserted, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* for a list of supported components.

value

Is the numeric value to use for the requested component. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

H*format*

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Inserting the Day Component Into a Date-Time Field

The following request inserts the day into the ADD_MONTH field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
INSERT_DAY/HYYMDS = HSETPT(ADD_MONTH, 'DAY', 28, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         ADD_MONTH            INSERT_DAY
------  ---------         ---------            ----------
1118    2000/06/26 05:45  2000/08/26 05:45:00  2000/08/28 05:45:00
1237    2000/02/05 03:30  2000/04/05 03:30:00  2000/04/28 03:30:00
```

HMIDNT: Setting the Time Portion of a Date-Time Field to Midnight

Use the following syntax in an expression to change the time portion of a date-time field to midnight (all zeroes). This function can be used for testing date-time fields for a given date.

```
HMIDNT (dtfield, {8|10}, 'Hformat')
```

where:

dtfield

Is date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

Hformat

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Setting the Time to Midnight

The following request sets the time portion of the TRANSDATE field to midnight in both the 24- and 12-hour systems:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_MID_24/HYYMDS  = HMIDNT(TRANSDATE, 8, 'HYYMDS');
TRANSDATE_MID_12/HYYMDSA = HMIDNT(TRANSDATE, 8, 'HYYMDSA');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         TRANSDATE_MID_24     TRANSDATE_MID_12
------  ---------         ----------------     ----------------
1118    2000/06/26 05:45  2000/06/26 00:00:00  2000/06/26 12:00:00AM
1237    2000/02/05 03:30  2000/02/05 00:00:00  2000/02/05 12:00:00AM
```

HDATE: Converting the Date Portion of a Date-Time Field to a Date Format

Use the following syntax in an expression to extract the date portion of a date-time field and convert it to a date format (number of days since the base date 1900/12/31).

```
HDATE (dtfield, 'dateformat')
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

dateformat

Is the USAGE format of the returned date field (for example, YMD), enclosed in single quotation marks.

Example Converting the Date Portion of the TRANSDATE Field to a Date Format

The following request converts the date portion of the TRANSDATE field to date format YYMD:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         TRANSDATE_DATE
------  ---------         --------------
1118    2000/06/26 05:45  2000/06/26
1237    2000/02/05 03:30  2000/02/05
```

HDTTM: Converting a Date field to a Date-Time Field

Use the following syntax in an expression to convert a date field to a date-time field. The time portion is set to midnight.

```
HDTTM (datefield, {8|10}, Hformat)
```

where:

datefield

Is the date value to be converted. You can supply the name of a date field, a date constant, or an expression that returns a date value.

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

H*format*

Is the USAGE format of the returned date-time value.

Example Converting a Date Field to a Date-Time Field

The following request converts the date field TRANSDATE_DATE to a date-time field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
DT2/HYYMDIA = HDTTM(TRANSDATE_DATE, 8, 'HYYMDIA');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         TRANSDATE_DATE  DT2
------  ---------         --------------  ---
1118    2000/06/26 05:45  2000/06/26      2000/06/26 12:00AM
1237    2000/02/05 03:30  2000/02/05      2000/02/05 12:00AM
```

HTIME: Converting the Time Portion of a Date-Time Field to a Number

Use the following syntax in an expression to convert the time portion of a date-time field to a numeric number of milliseconds (if the first argument is 8) or microseconds (if the first argument is 10). For microseconds, the input date-time field must be a 10-byte field.

```
HTIME ({8|10}, dtfield, 'Dformat')
```

where:

`8 | 10`

Is the length of the input date-time value. Use 8 for time values down to milliseconds, 10 for input time values down to microseconds.

dtfield

Is the date-time value to use for extracting the time. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

Dformat

Is the USAGE format of the returned number of milliseconds or microseconds, enclosed in single quotation marks.

Example Converting the Time Portion of a Date-Time Field to a Number

The following request converts time portion of the TRANSDATE field to a number of milliseconds:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
MILLISEC/D12.2 = HTIME(8, TRANSDATE, 'D12.2');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME               MILLISEC
------  ---------               --------
1118    2000/06/26 05:45   20,700,000.00
1237    2000/02/05 03:30   12,600,000.00
```

HGETC: Storing the Current Date and Time in a Date-Time Field

Use the following syntax in an expression store the current date and time in a date-time field. If millisecond or microsecond values are not available in your operating environment, the value returned for these components is zero.

```
HGETC ({8|10}, 'Hformat')
```

where:

`8 | 10`

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for input time values down to microseconds.

`Hformat`

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Storing the Current Date and Time in a Date-Time Field

The following request stores the current date and time in field DT2:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DT2/HYYMDm = HGETC(10, 'HYYMDm');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         DT2
------  ---------         ---
1118    2000/06/26 05:45  2000/10/03 15:34:24.000000
1237    2000/02/05 03:30  2000/10/03 15:34:24.000000
```

NF759: Increased Number of Display Fields

The maximum number of display fields allowed in a request has been increased from 256 to approximately 495. This number includes all named fields, whether printed or not, such as data source fields, temporary fields (virtual fields and calculated values), certain internal fields (for example, TABPAGENO) and fields used in headings and footings. The increased field limit is 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.

Reference Error Messages for Number of Display Fields

(FOC005) **THE NUMBER OF VERB OBJECTS EXCEEDS THE MAXIMUM**

Up to 495 fields may be used in a single report request. This total does not include sort fields.

(FOC098) **TOTAL LENGTH OF VERB OBJECTS EXCEED 3956 CHARACTERS**

The length of all verb objects was greater than 3956. Adjust the lengths or quantity of fields and reissue the TABLE request.

Reference Usage Notes for Number of Display Fields

- Out of a 4K there exists a 3956 byte data area. (The data area is where fields are stored for report creation.)
- Every field in this data area is rounded up to a full word boundary.
- Every field gets a four-byte counter field associated with it and the four-byte counter field affects the total number of bytes in this data area.

- Field Prefixes, and formatting options may also affect the available data area.

NF755: Using FILEDEF for Creating Extract Files

By default in VM/CMS, extract files are written to the VM minidisk specified by the SET TEMP command. If you do not issue the SET TEMP command, extract files are written to the minidisk with the largest amount of unused space to which you have write access. The name of an extract file is the AS name specified in the command that creates it or, if no AS name is specified, a default name (HOLD, SAVE, orSAVB). The file type is assigned based on the extract file format.

This feature enables you to use a FILEDEF command to assign a file name, file type, and file mode for an extract file. In prior releases, FILEDEF was supported for creating SAVE and SAVB files, but not for creating HOLD files.

In the case of a HOLD file, the Master File is not affected by the FILEDEF command. The Master File is written to the minidisk specified by the SET TEMP command, and its name is taken from the AS name in the HOLD command. If the HOLD command does not contain an AS phrase, the Master File name is HOLD.

The HOLD, SAVE, or SAVB command can be issued within a request or after a request executes.

Note: The FILEDEF command is not supported on OS/390 or MVS. As in prior releases, you can use the DYNAM ALLOCATE or TSO ALLOCATE command to dynamically allocate extract files.

Syntax How to Use a FILEDEF Command for Creating an Extract File

Issue the following command before creating the extract file:

```
CMS FILEDEF ddname DISK filename filetype filemode
```

where:

ddname

Is the AS name from the HOLD, SAVE, or SAVB command. If the command did not specify an AS name, the ddname is HOLD, SAVE, or SAVB.

filename

Is the file name for the extract file.

filetype

Is the file type for the extract file.

filemode

Is the file mode for the extract file. You must have write access to this minidisk. If you do not have write access, the following error message is returned:

```
(FOC350) ERROR WRITING OUTPUT FILE: filename
```

Note:

- If a FOCSORT file is created, it is written to the minidisk specified by the SET TEMP command.
- The FILEDEF command must be in effect any time you use FOCUS to access the file.
- Do not specify DCB parameters for a HOLD file; if you do, they will be ignored.

- The FILEDEF command is not supported for creating extract files in FOCUS format or other DBMS formats.
- The FILEDEF command is supported for MATCH FILE requests.

Example Using FILEDEF to Create a HOLD File

In the following examples, SET TEMP = T. The request is:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY LAST_NAME BY FIRST_NAME
ON TABLE HOLD AS CURRSAL
END
```

Running this request with no FILEDEF command creates the following files:

```
CURRSAL  MASTER   T1
CURRSAL  FOCTEMP  T1
```

Issue the following FILEDEF command:

```
CMS FILEDEF CURRSAL DISK SALLIST DATA A
```

Now, running the same request creates the following files:

```
CURRSAL  MASTER   T1
SALLIST  DATA     A1
```

Note that the file name, file type, and destination minidisk for the extract file are taken from the FILEDEF command. The file name and destination minidisk for the Master File are not.

NF761: Comma Suppress Edit Format Option

The comma suppress edit format option allows you to suppress the display of commas. This gives you the ability to display numeric and monetary data without commas and gives you an additional display option when using COMPUTE/DEFINE.

Syntax How to Suppress Commas

The comma suppress edit option is invoked by including a lower case ‘c’ after numeric format options M and N (floating and non-floating dollar sign) and data format D (floating-point double-precision).

Format	Data	Display
D6c	41376	41376
D7Mc	6148	$6148
D7Nc	6148	$ 6148

Reference Error Messages for Comma Suppress Option

Using the comma suppress edit option with alphanumeric data types causes the following error message to be displayed:

```
(FOC207) ERROR IN THE FORMAT DEFINITION OF FIELD
```

Reference Considerations for Comma Suppress Option

This feature may only be used with numeric format options that automatically invoke comma inclusion option C.

Using this option with other numeric data types has no effect.

NF762: Percent Edit Format Option

FOCUS provides a format option that enables you to display a percent sign along with numeric data. This allows you to display data as percentages after calculations using COMPUTE/DEFINE.

Syntax How to Display Percent Signs

The percent edit option is invoked by including a '%' after any numeric format option.

Format	Data	Display
I2%	21	21%
D7%	6148	6,148%
F3.2%	48	48.00%

Reference Error Messages for Percent Display Option

Using the percent edit format option with either alphanumeric data types, format options M and N, or date formats results in the following error message being displayed:

```
(FOC207) ERROR IN THE FORMAT DEFINITION OF FIELD
```

Reference Special Considerations for Percent Display Option

This feature may only be used with numeric data.

This feature does not perform calculations intended to derive a percentage of data. It is only a format option that allows numeric data to be displayed as a percentage.

The percent sign may only be displayed on the right hand side of the data.

NF766: DEFINE Functions

A DEFINE function is a named group of calculations that use any number of input values and produce a return value. These functions are not tied to a specific Master File or request and therefore can be reused in different contexts throughout your FOCUS session.

Using DEFINE Functions

A DEFINE function can be called in the same situations that are valid for user-written subroutines. Data types (numeric or alphanumeric) must match between the arguments defined and the arguments used. Shorter alphanumeric arguments are padded with blanks while longer alphanumeric arguments are truncated.

All calculations within the function are done in double precision. As with other FOCUS calculations, format conversions occur only across equal signs in the assignments that define temporary fields.

Before calling a DEFINE function, you must issue the commands that define the function at the FOCUS prompt or in a stored procedure.

Syntax How To Define a Function

```
DEFINE FUNCTION name (parameter1/format1,..., parametern/formatn)
[tempvariablea/formata = expressiona;]
   .
   .
   .
[tempvariablex/formatx = expression;x]
name/format = [result_expression];
END
```

where:

name

Is the name of the function. This must be the last field calculated in the function and is used to return the value of the function to the calling procedure.

format

Is the format of the value the function returns.

parameter1/format1...parametern/formatn

Are the parameter names and their formats.

If a parameter is alphanumeric, the calling argument must be alphanumeric. Shorter calling arguments are padded on the right with blanks, and longer arguments are truncated.

If a parameter is numeric, the calling argument must also be numeric. To prevent unexpected results, you must be consistent in your use of data types.

tempvariablea/formata...tempvariablex/formatx

Are temporary fields and their formats. Temporary fields hold intermediate values used in the function. You can define as many temporary fields as you need.

expressiona...expressionx

Are the expressions that calculate the temporary field values. The expressions can use parameters, constants, and other temporary fields defined in the same function.

result_expression

Is the expression that calculates the value returned by the function. The expression can use parameters, constants, and temporary fields defined in the same function.

All names defined in the body of the function are local to the function. The last field defined before the END command in the function definition must have the same name as the function and represents the return value for the function.

Example Defining a Function

```
DEFINE FUNCTION SUBTRACT (VAL1/D8, VAL2/D8)
SUBTRACT/D8.2 = VAL1 - VAL2;
END

TABLE FILE MOVIES
PRINT TITLE LISTPR IN 35 WHOLESALEPR AND COMPUTE
PROFIT/D8.2 = SUBTRACT(LISTPR,WHOLESALEPR);
BY CATEGORY
  WHERE CATEGORY EQ 'MYSTERY' OR 'ACTION'
END
```

SUBTRACT is the name of the function. It uses local parameters VAL1 and VAL2.

The output is:

```
CATEGORY  TITLE                    LISTPR  WHOLESALEPR     PROFIT
--------  -----                    ------  -----------     ------
ACTION    JAWS                      19.95        10.99       8.96
          ROBOCOP                   19.98        11.50       8.48
          TOTAL RECALL              19.99        11.99       8.00
          TOP GUN                   14.95         9.99       4.96
          RAMBO III                 19.95        10.99       8.96
MYSTERY   REAR WINDOW               19.98         9.00      10.98
          VERTIGO                   19.98         9.00      10.98
          FATAL ATTRACTION          29.98        15.99      13.99
          NORTH BY NORTHWEST        19.98         9.00      10.98
          DEAD RINGERS              25.99        15.99      10.00
          MORNING AFTER, THE        19.95         9.99       9.96
          PSYCHO                    19.98         9.00      10.98
          BIRDS, THE                19.98         9.00      10.98
          SEA OF LOVE               59.99        30.00      29.99
```

Syntax How to Query DEFINE Functions

You can display a list of all defined functions and their parameters by issuing:

```
? FUNCTION
```

A screen similar to the following is displayed:

```
FUNCTIONS CURRENTLY ACTIVE

Name      Format    Parameter     Format
--------  --------  ------------  --------
SUBTRACT  D8.2      VAL1          D8
                    VAL2          D8
```

If you issue the ? FUNCTION command with no functions defined, the following is displayed:

```
NO FUNCTIONS CURRENTLY IN EFFECT
```

Syntax How to Clear DEFINE Functions

You can clear functions by issuing:

```
DEFINE FUNCTION {name|*} CLEAR
```

where:

name

Is the function name to clear.

*

Clears all active functions.

Reference DEFINE Function Limits and Restrictions

- The number of functions you can define and use in a session is virtually unlimited.
- DEFINE functions are not cleared by issuing a JOIN, or any other FOCUS command, with the exception of DEFINE FUNCTION CLEAR.
- Function names are limited to eight characters. There is no limit on the number of parameters.
- Parameter names are limited to twelve characters.
- DEFINE functions can use other DEFINE functions but cannot be used recursively.

- If you overwrite or clear a DEFINE function, a subsequent attempt to use a temporary field that refers to that function will generate the following message:

  (FOC1956) DEFINE FUNCTION %1 used after CLEAR or reDEFINE

  DEFINE FUNCTION dfname was invoked, but it no longer exists. A "DEFINE FUNCTION dfname" command was processed subsequently (with or without CLEAR), thereby invalidating any references to the the original version of dfname.

Reference DEFINE FUNCTION Error Messages

(FOC1940) **DEFINE FUNCTION name not 1-8 characters: %1**

The name given for a DEFINE FUNCTION (the character string following "DEFINE FUNCTION") contains more than eight characters or is null.

(FOC1941) **DEFINE FUNCTION %1 CLEAR--illegal delimiter**

In a "DEFINE FUNCTION dfname CLEAR" command, the character string dfname contains a forward slash, an open or close parenthesis, or a comma; this is not supported.

(FOC1942) **DEFINE FUNCTION not found: %1**

In a "DEFINE FUNCTION dfname CLEAR" command, the syntax is proper but dfname does not exist, so it cannot be CLEARed.

(FOC1943) **DEFINE FUNCTION needs close paren to delimit params: %1**

A DEFINE FUNCTION definition must contain a list of parameters delimited by a close parenthesis. The DEFINE FUNCTION parser encountered END or a similar termination without finding a close parenthesis.

(FOC1944) **No open paren after DEFINE FUNCTION format: %1**

A format may be specified for a DEFINE FUNCTION immediately after the DEFINE FUNCTION name (and a forward slash) and before the open parenthesis enclosing the parameter list. The forward slash is coded but the next delimiter is not an open parenthesis.

(FOC1945) **DEFINE FUNCTION format not 1-8 characters: %1**

A format may be specified for a DEFINE FUNCTION immediately after the DEFINE FUNCTION name (and a forward slash) and before the open parenthesis enclosing the parameter list. The length of the format specification must be from one to eight characters.

(FOC1946) **Name of DEFINE FUNCTION not delimited by "/" or "(": %1**

The delimiter for the DEFINE FUNCTION name must be a forward slash (if a format specification follows) or an open parenthesis (if there is no format specification). The actual delimiter is neither of these.

(FOC1947) **DEFINE FUNCTION parameter name not delimited by "/": %1**

A format specification must follow each parameter for a DEFINE FUNCTION, and the delimiter that separates the name from the format specification must be a forward slash. The actual delimiter is not a forward slash.

(FOC1948) **DEFINE FUNCTION parameter name not 1-12 characters: %1**

The name of each parameter for a DEFINE FUNCTION must contain from one to twelve characters. Either the parameter name shown is null or it contains more than twelve characters.

(FOC1949) **DEFINE FUNCTION parameter format not delimited by "," or ")": %1**

A format specification must follow each parameter for a DEFINE FUNCTION, and the delimiter that marks the end of the specification must be a comma (if there are additional parameters) or a close parenthesis (if this is the last parameter).

(FOC1950) **DEFINE FUNCTION parameter format not 1-8 characters: %1**

A format specification must follow each parameter for a DEFINE FUNCTION, and it must contain from one to eight characters. Either the actual specification contains more than eight characters or it is null.

(FOC1951) **Invalid DEFINE FUNCTION parameter format: %1**

The format specification given for the parameter shown for a DEFINE FUNCTION is not recognized.

(FOC1952) **DEFINE FUNCTION %1 is void**

At least one computational statement must appear between the parameter list for DEFINE FUNCTION and "END", but the actual DEFINE FUNCTION has no computational statements.

(FOC1953) **DEFINE FUNCTION name %1 not equal to final assignment name %2**

The name immediately following "DEFINE FUNCTION" must match the name on the left-hand side of the final computational statement given for the DEFINE FUNCTION, but it does not match.

(FOC1954) **DEFINE FUNCTION format %1 not equal to final assignment format %2**

The (optional) format specified before the parameters for a DEFINE FUNCTION must match the format on the left-hand side of the final computational statement given for the DEFINE FUNCTION, but it does not match.

(FOC1955) **DEFINE FUNCTION %1 entered recursively**

A DEFINE FUNCTION was invoked, but before returning it was invoked again; this is not supported. Note that the invalid recursive usage may have been indirect: DF1 invokes DF2, which invokes DF3, ... , which invokes DF1.

(FOC1956) **DEFINE FUNCTION %1 used after CLEAR or reDEFINE**

DEFINE FUNCTION dfname was invoked, but it no longer exists. A "DEFINE FUNCTION dfname" command was processed subsequently (with or without CLEAR), thereby invalidating any references to the original version of dfname.

(FOC1957) **Warning--DEFINE FUNCTION %1 superseded; could result in FOC1956**

A successful DEFINE FUNCTION dfname has replaced the prior dfname. Trying to use a DEFINE FILE (or another DEFINE FUNCTION) that refers to the prior version of dfname will result in FOC1956. Such a DEFINE FILE/FUNCTION could be re-issued, which would direct it to use the new version of dfname.

(FOC1958) **DEFINE FUNCTION %1 takes %2 parameter(s), but # supplied**

DEFINE FUNCTION dfn requires Nexp input parameters, but Nsupp were supplied. dfn, Nexp, and Nsupp are displayed within the message text.

NF773: Token Delimited Files

FOCUS can now read single-segment sequential data sources in which fields are separated by any delimiter.

Syntax How to Define Files With Delimiters

Delimiters must be defined in the Master File. The FILE declaration must include the following attribute:

```
SUFFIX=DFIX
```

To use a delimiter that consists of a single non-printable character or of one or more printable characters, the delimiter is defined as a field with the following attributes:

```
FIELDNAME=DELIMITER, ALIAS=delimiter, USAGE=ufmt, ACTUAL=afmt ,$
```

To use a delimiter that consists of multiple non-printable characters or a combination of printable and non-printable characters, the delimiter is defined as a group:

```
GROUP=DELIMITER,      ALIAS=          , USAGE=ufmtg, ACTUAL=afmtg ,$
 FIELDNAME=DELIMITER, ALIAS=delimiter1, USAGE=ufmt1, ACTUAL=afmt1 ,$
   .
   .
   .
 FIELDNAME=DELIMITER, ALIAS=delimitern, USAGE=ufmtn, ACTUAL=afmtn ,$
```

where:

DELIMITER

Indicates that the field or group is used as the delimiter in the data source.

delimiter

Identifies a delimiter.

For one or more printable characters, the value consists of the actual characters. The delimiter must be enclosed in single quotation marks if it includes characters used as delimiters in Master File syntax.

For a non-printable character, the value is the decimal equivalent of the EBCDIC or ASCII representation of the character, depending on your operating environment.

ufmt, afmt

Are the USAGE and ACTUAL formats for the delimiter. Possible values are:

Type of delimiter	USAGE	ACTUAL
Printable characters	*An* where *n* is the number of characters	*An* where *n* is the number of characters
Non-printable character such as Tab	I4	I1
Group (combination of printable and non-printable characters, or multiple non-printable characters)	Sum of the individual USAGE lengths	Sum of the individual ACTUAL lengths

Reference Usage Notes for Token Delimited Files

- If the delimiter is alphanumeric and the delimiter value contains special characters (those used as delimiters in Master File syntax), it must be enclosed in single quotation marks.
- Numeric (decimal) values may be used to represent any character, but are predominantly used for non-printable characters such as Tab. The numeric values may differ between EBCDIC and ASCII platforms.
- A delimiter is needed to separate field values. A pair of delimiters denotes a missing or default field value.
- Trailing delimiters are not necessary except that all fields must be terminated with the delimiter if the file resides in CMS or has fixed length records in MVS.
- Only one delimiter field/group is permitted per Master File.
- Token delimited files cannot be used in joins.

Example Defining Delimiters

The following example shows a one-character alphanumeric delimiter:

```
FIELDNAME=DELIMITER, ALIAS=',' ,USAGE=A1, ACTUAL=A1  ,$
```

The following example shows a two-character alphanumeric delimiter:

```
FIELDNAME=DELIMITER, ALIAS=//  ,USAGE=A2, ACTUAL=A2  ,$
```

The following example shows how to use the Tab character as a delimiter:

```
FIELDNAME=DELIMITER, ALIAS=05  ,USAGE=I4, ACTUAL=I1  ,$
```

The following example shows how to use a blank character described as a numeric delimiter:

```
FIELDNAME=DELIMITER, ALIAS=64  ,USAGE=I4, ACTUAL=I1  ,$
```

The following example shows a group delimiter (Tab-slash-Tab combination):

```
GROUP=DELIMITER,      ALIAS=    ,USAGE=A9, ACTUAL=A3  ,$
FIELDNAME=DEL1,       ALIAS=05  ,USAGE=I4, ACTUAL=I1  ,$
FIELDNAME=DEL2,       ALIAS=/   ,USAGE=A1, ACTUAL=A1  ,$
FIELDNAME=DEL3,       ALIAS=05  ,USAGE=I4, ACTUAL=I1  ,$
```

Example Separating Field Values for Missing Data

The following Master File shows the MISSING attribute specified for the CAR field:

```
FILE=DFIXF01  ,SUFFIX=DFIX
SEGNAME=SEG1  ,SEGTYPE=S0
FIELDNAME=COUNTRY   ,ALIAS=F1  ,USAGE=A10 ,ACTUAL=A10  ,$
FIELDNAME=CAR       ,ALIAS=F2  ,USAGE=A16 ,ACTUAL=A16  ,MISSING=ON, $
FIELDNAME=NUMBER    ,ALIAS=F3  ,USAGE=P10 ,ACTUAL=Z10  ,$
FIELDNAME=DELIMITER ,ALIAS=',' ,USAGE=A1  ,ACTUAL=A1   ,$
```

In the source file, two consecutive comma delimiters indicate missing values for CAR:

```
GERMANY,VOLKSWAGEN,1111
GERMANY,BMW,
USA,CADILLAC,22222
USA,FORD
USA,,44444
JAPAN
ENGLAND,
FRANCE
```

The output is:

COUNTRY	CAR	NUMBER
GERMANY	VOLKSWAGEN	1111
GERMANY	BMW	0
USA	CADILLAC	22222
USA	FORD	0
USA	.	44444
JAPAN	.	0
ENGLAND		0
FRANCE	.	0

NF777: Two-Gigabyte FOCUS Database Support

The FOCUS database size has been increased to a maximum of two gigabytes per physical data file, overcoming previous FOCUS database size limitations. Through partitioning, one logical FOCUS database can now span up to 500 gigabytes. FOCUS database size was previously limited to 64 gigabytes through the use of LOCATION files.

Note that the discussion in this document applies to any FOCUS file that has extended beyond one-gigabyte but has not reached the two-gigabyte limit.

For information about partitioning FOCUS data sources, see *NF777: Partitioned FOCUS Data Sources*.

Enabling Two-Gigabyte Support

In order to enable support for two-gigabyte databases, you need to set the value of the FOC2GIGDB parameter to ON in the FOCPARM profile.

Syntax How to Enable Two-Gigabyte Support

Issue the following command in the FOCPARM profile:

```
SET FOC2GIGDB = {ON|OFF}
```

where:

`ON`

Enables support for FOCUS data sources larger than one-gigabyte. Note that an attempt to use FOCUS data sources larger than one-gigabyte in a release prior to FOCUS Version 7.1 can cause database corruption.

`OFF`

Disables support for FOCUS data sources larger than one-gigabyte. OFF is the default value.

Reference Usage Notes for Two-Gigabyte FOCUS Data Sources

- To sort a FOCUS data source that is larger than one-gigabyte, on MVS you must explicitly allocate ddname FOCSORT to a temporary file with enough space to contain the data; on VM, you must have enough TEMP space available.
- To REBUILD a FOCUS data source that is larger than one-gigabyte, on MVS you must explicitly allocate ddname REBUILD to a temporary file with enough space to contain the data; on VM you must have enough TEMP space available. It is strongly recommended that you REBUILD/REORG to a new file, in sections, to avoid the need to allocate large amounts of space to REBUILD. In the DUMP phase, use selection criteria to dump a section of the database. In the LOAD phase, make sure to *add* each new section after the first. To add to a database in MVS you must issue the LOAD command with the following syntax:

  ```
  LOAD NOCREATE
  ```

- If you create a FOCUS data source that is larger than one gigabyte using HOLD FORMAT FOCUS, on MVS you must explicitly allocate ddnames FOC$HOLD and FOCSORT to temporary files large enough to hold the data; on VM you must have enough TEMP space available.

NF777: Partitioned FOCUS Data Sources

Through partitioning, one logical FOCUS database can now span up to 500 gigabytes. FOCUS database size was previously limited to 64 gigabytes through the use of LOCATION files.

For information about two-gigabyte support, see *NF777: Two-Gigabyte FOCUS Database Support*

Partitioning

FOCUS data sources can now consist of up to 250 physical files of up to two gigabytes each, for a maximum of 500 gigabytes of real storage per logical database. The new horizontal partition is a slice of the entire file structure, meaning that new FOCUS database partitions are not subject to the multiplicative effect of LOCATION-style vertical partitioning. Note, however, that the number of physical files associated with one FOCUS data source is the sum of all of its partitions and LOCATION files. This sum must be less than or equal to 250. New FOCUS data sources can grow in size over time, and can be repartitioned based on the requirements of the application.

Note: You do not have to partition your data source. If you choose not to, your application will automatically support the increased FOCUS database size when you set the FOC2GIGDB parameter to ON.

Intelligent Partitioning

The FOCUS database now supports intelligent partitioning, which means that each horizontal partition contains the complete database structure for specific data values or ranges of values. Intelligent partitioning lets you not only separate the data into up to 250 physical two-gigabyte files, it allows you to create an Access File in which you describe, using WHERE criteria, the actual data values in each partition. When processing a report request, the selection criteria in the request are compared to the WHERE criteria in the Access File to determine which partitions are required for retrieval.

To select applications that can benefit most from partitioning, look for applications that employ USE commands to concatenate data sources or for data that lends itself to separation based on data values or ranges of values, such as data stored by month or by department. Intelligent partitioning functions like an intelligent USE. It looks at the Access File when processing a report request to determine which partitions to read, whereas the USE command reads all of the files on the list. This intelligence decreases I/O and delivers significant performance benefits.

To take advantage of the partitioning feature, you must:

- Edit the Master File and add the ACCESSFILE attribute.
- Create the Access File using a text editor.

Concatenation of multiple partitions is supported for reporting only. You must load or rebuild each physical partition separately.

You can either create a separate Master File for each partition to reference in the load procedure, or you can use the single Master File created for reporting against the partitioned data source, if you:

- Issue an explicit allocation command to link the Master File to each partition in turn.
- Run the load procedure for each partition in turn.

Note: Report requests will automatically read all required partitions without user intervention.

Specifying an Access File in a FOCUS Master File

To take advantage of the partitioning feature, you must edit the Master File and add the ACCESSFILE attribute to identify the name of the Access File.

Syntax How to Specify an Access File for a Partitioned FOCUS Database

```
FILENAME=fname, SUFFIX=FOC, ACCESS[FILE]=accessfile,
.
.
.
```

where:

fname

Is the file name of the partitioned data source.

accessfile

Is the name of the Access File. Note that this can be any valid name.

Example Master File for the VIDEOTR2 Partitioned Database

```
FILENAME=VIDEOTR2,  SUFFIX=FOC,
ACCESS=VIDEOACX,  $
SEGNAME=CUST,       SEGTYPE=S1
 FIELDNAME=CUSTID,       ALIAS=CIN,          FORMAT=A4,       $
 FIELDNAME=LASTNAME,     ALIAS=LN,           FORMAT=A15,      $
 FIELDNAME=FIRSTNAME,    ALIAS=FN,           FORMAT=A10,      $
 FIELDNAME=EXPDATE,      ALIAS=EXDAT,        FORMAT=YMD,      $
 FIELDNAME=PHONE,        ALIAS=TEL,          FORMAT=A10,      $
 FIELDNAME=STREET,       ALIAS=STR,          FORMAT=A20,      $
 FIELDNAME=CITY,         ALIAS=CITY,         FORMAT=A20,      $
 FIELDNAME=STATE,        ALIAS=PROV,         FORMAT=A4,       $
 FIELDNAME=ZIP,          ALIAS=POSTAL_CODE,  FORMAT=A9,       $
SEGNAME=TRANSDAT, SEGTYPE=SH1,  PARENT=CUST
 FIELDNAME=TRANSDATE,    ALIAS=OUTDATE,   FORMAT=HYYMDI,        $
SEGNAME=SALES,    SEGTYPE=S2,   PARENT=TRANSDAT
 FIELDNAME=TRANSCODE,    ALIAS=TCOD,    FORMAT=I3,            $
 FIELDNAME=QUANTITY,     ALIAS=NO,      FORMAT=I3S,           $
 FIELDNAME=TRANSTOT,     ALIAS=TTOT,    FORMAT=F7.2S,         $
SEGNAME=RENTALS,  SEGTYPE=S2,   PARENT=TRANSDAT
 FIELDNAME=MOVIECODE,    ALIAS=MCOD,      FORMAT=A6, INDEX=I, $
 FIELDNAME=COPY,         ALIAS=COPY,      FORMAT=I2,          $
 FIELDNAME=RETURNDATE,   ALIAS=INDATE,    FORMAT=YMD,         $
 FIELDNAME=FEE,          ALIAS=FEE,       FORMAT=F5.2S,       $
DEFINE DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
```

The FOCUS Access File

The Access File provides comprehensive metadata management for all FOCUS data sources. It shields end users from the complex file storage and configuration details used for efficient and transparent access to partitioned and distributed databases.

The Access File describes how to locate, concatenate, join, and select the appropriate physical data files for retrieval requests against one or more FOCUS databases. Access Files are optional in retrieval requests against non-partitioned databases with no location files and play no part in data maintenance requests.

Every request supplies the name of a Master File. The Master File is read and the declarations in it are used to access the data source. If the Master File includes an ACCESSFILE attribute, FOCUS reads the named Access File and uses it to locate the correct data files. Each Master File can point to its own separate Access File, or several Master Files can point to the same Access File. This flexibility makes it possible to create one Access File that manages database access for an entire application. If the Master File does not contain an ACCESSFILE attribute, FOCUS attempts to satisfy the request with the Master File alone.

You can use an Access File to take advantage of the following database features:

- Horizontal and vertical partitioning. A database can consist of several separate files, or horizontal partitions, each of which contains the database records for a specific time period, region, or other element. Segments can also be stored separately from the rest of the data source (LOCATION files or vertical partitions). The Access File describes how to concatenate the separate data files.
- Joins. If joined files are partitioned, the Access File describes how to concatenate the separate data files in the join.

An Access File is *required* to take advantage of intelligent partitioning. Intelligent partitioning places specific data values in each physical partition and uses the Access File to describe the values in each partition. With this information, FOCUS optimizes database access by retrieving only those partitions whose values are consistent with the selection criteria in the request.

Note: On OS/390 the Access File must be a member of a data set concatenated in the allocation for ddname ACCESS. On VM/ESA the Access File must have the file type ACCESS. FOCSQL cannot be used as the file type. The Access File has the same DCB attributes as the Master File (LRECL=80, RECFM=FB, BLKSIZE= multiple of LRECL).

FOCUS Access File Attributes

The Access File can include the following attributes:

Attribute	Synonyms	Description
MASTERNAME	MASTER	A Master File entry.
DATANAME	DATA	The name of the physical file.
WHERE		The WHERE criteria.
LOCATION		A segment location.

Each Access File declaration begins with a MASTERNAME attribute that identifies the Master File to which it applies. By including multiple MASTERNAME declarations, you can use one Access File for multiple Master Files, possibly for an entire application.

Syntax How to Create an Access File

```
MASTERNAME filename1
 DATANAME dataname1 [WHERE test1 ;]
  [LOCATION locationnamea DATANAME datanamea]
   .
   .
   .
 DATANAME dataname2 [WHERE test2;]
  [LOCATION locationnameb DATANAME datanameb]
   .
   .
   .
MASTERNAME filename2
   .
   .
   .
```

where:

`MASTERNAME`

Is the attribute that identifies the Master File name. MASTER is a synonym for MASTERNAME.

filename1, filename2

Are names of Master Files. You can describe unrelated Master Files in one Access File.

`DATANAME`

Is the attribute that identifies a physical file. DATA is a synonym for DATANAME.

dataname1, dataname2

Are the fully qualified physical file names of physical partition files, in the syntax native to your operating environment.

test

Is a valid WHERE test. The following types of expressions are supported. You can also combine any number of these expressions with the AND operator:

fieldname relational_operator value1 [OR *value2* OR *value3* ...]

fieldname FROM *value1* TO *value2* [OR *value3* TO *value4* ...]

fieldname1 FROM *value1* TO *value2* [OR *fieldname2* FROM *value3* TO *value4* ...]

where:

fieldname, *fieldname1*, *fieldname2*

Are field names in the Master File.

relational_operator

Can be one of the following: EQ, NE, GT, GE, LT, LE.

value1, *value2*, *value3*, *value4*

Are valid values for their corresponding fields.

Note: If the test conditions do not accurately reflect the contents of the files, you may get incorrect results from requests.

LOCATION

Is the attribute that identifies a separately stored segment.

locationnamea,locationnameb

Are the values of the LOCATION attributes from the Master File.

Segment locations must map one-to-one to horizontal partitions.

datanamea, datanameb

Are the fully qualified physical file names of the LOCATION files, in the syntax native to your operating environment.

Example Describing Intelligent Partitions in a FOCUS Access File

The following Access File illustrates how to define intelligent partitions for the VIDEOTR2 database, in which data is grouped by date.

For MVS:

```
MASTERNAME VIDEOTR2
  DATANAME USER1.VIDPART1.FOCUS
     WHERE DATE EQ 1991;

  DATANAME USER1.VIDPART2.FOCUS
     WHERE DATE FROM 1996 TO 1998;

  DATANAME USER1.VIDPART3.FOCUS
     WHERE DATE FROM 1999 TO 2000;
```

For CMS:

```
MASTERNAME VIDEOTR2
  DATANAME 'VIDPART1 FOCUS A'
     WHERE DATE EQ 1991;

  DATANAME 'VIDPART2 FOCUS A'
     WHERE DATE FROM 1996 TO 1998;

  DATANAME 'VIDPART3 FOCUS A'
     WHERE DATE FROM 1999 TO 2000;
```

Example Describing Intelligent Partitions With LOCATION Files

Consider the following version of a SALES Master File. The CUSTDATA segment is stored in a separate LOCATION file named MORECUST:

```
FILENAME=SALES, ACCESSFILE=XYZ,$
  SEGNAME=SALEDATA
  .
  .
  .
  SEGNAME=CUSTDATA, LOCATION=MORECUST,$
```

The corresponding Access File (XYZ) describes one partition for 1994 data, and another partition for the 1993 data. Each partition has its corresponding MORECUST LOCATION file:

For MVS:

```
MASTERNAME SALES
 DATANAME USER1.SALES94.FOCUS
    WHERE SDATE FROM '19940101' TO '19941231';
   LOCATION MORECUST
     DATANAME USER1.MORE1994.FOCUS

 DATANAME USER1.SALES93.FOCUS
    WHERE SDATE FROM '19930101' TO '19931231';
   LOCATION MORECUST
     DATANAME USER1.MORE1993.FOCUS
```

For CMS:

```
MASTERNAME SALES
 DATANAME 'SALES94  FOCUS A'
    WHERE SDATE FROM '19940101' TO '19941231';
   LOCATION MORECUST
     DATANAME 'MORE1994 FOCUS A'

 DATANAME 'SALES93  FOCUS A'
    WHERE SDATE FROM '19930101' TO '19931231';
   LOCATION MORECUST
     DATANAME 'MORE1993 FOCUS A'
```

Example Using a Partitioned Database

The following illustrates how to use a partitioned database.

```
TABLE FILE VIDEOTR2
PRINT LASTNAME FIRSTNAME DATE
WHERE DATE FROM 1996 TO 1997
END
```

The output is:

```
LASTNAME         FIRSTNAME   DATE
--------         ---------   ----
HANDLER          EVAN        1996
JOSEPH           JAMES       1997
HARRIS           JESSICA     1997
HARRIS           JESSICA     1996
MCMAHON          JOHN        1996
WU               MARTHA      1997
CHANG            ROBERT      1996
```

There is nothing in the request or output that signifies that a partitioned database was used. However, only the second partition is retrieved, reducing I/O and enhancing performance.

Describing Joined Files

The Master File can describe cross-references to other Master Files. In simple cases, the Master File alone may be sufficient for describing the cross-reference.

If one of the joined files is horizontally partitioned, only that file needs an Access File to implement the join.

However, when both of the joined files are horizontally partitioned, they can both be described in one Access File or they can each be described in a separate Access File in order to implement the join. Only the host file is allowed to have WHERE criteria in the Access File. If both the host and cross-referenced file have WHERE criteria, a join may produce unexpected results.

Example Joining Two Partitioned Data Sources

Recall that the cross-referenced field in a join must be indexed. If the host file is partitioned, the cross-referenced file must either contain the same number of partitions as the host file or only one partition.

For MVS:

```
MASTERNAME SALES
 DATANAME USER1.NESALES.FOCUS
 DATANAME USER1.MIDSALES.FOCUS
 DATANAME USER1.SOSALES.FOCUS
 DATANAME USER1.WESALES.FOCUS

MASTERNAME CUSTOMER
 DATANAME USER1.NECUST.FOCUS
 DATANAME USER1.MIDCUST.FOCUS
 DATANAME USER1.SOCUST.FOCUS
 DATANAME USER1.WECUST.FOCUS
```

For CMS:

```
MASTERNAME SALES
 DATANAME 'NESALES  FOCUS A'
 DATANAME 'MIDSALES FOCUS A'
 DATANAME 'SOSALES  FOCUS A'
 DATANAME 'WESALES  FOCUS A'

MASTERNAME CUSTOMER
 DATANAME 'NECUST  FOCUS A'
 DATANAME 'MIDCUST FOCUS A'
 DATANAME 'SOCUST  FOCUS A'
 DATANAME 'WECUST  FOCUS A'
```

Reference Usage Notes for Partitioned FOCUS Data Sources

- Concatenation of multiple partitions in one request is only valid for reporting. To MODIFY or REBUILD a partitioned database, you must explicitly allocate and MODIFY, Maintain, or REBUILD one partition at a time.
- The order of precedence for allocating data sources is as follows:
  - A USE command in effect has the highest precedence. It overrides an Access File or an explicit allocation for a data source.
  - An Access File overrides an explicit allocation for a data source.
- A DATASET attribute cannot be used in the same Master File as an ACCESSFILE attribute.

Reference Error Messages

(ACC20201) **BOTH DATASET AND ACCESS FILE NOT PERMITTED**

A Master file cannot specify a DATASET and an Access File at the same time.

(ACC20202) **MEMORY ALLOCATION ERROR**

Access File Memory Failure.

(ACC20203) **ACCESS FILE %1 (MASTER %2) NOT FOUND**

The Access File specified in the Master could not be found. Check the spelling and/or the location of the Access File.

(ACC20204) **ACCESS FILE SYSTEM ERROR**

The Access File system could not execute the Access File.

(ACC20205) **UNABLE TO OPEN ACCESS FILE %1**

The Access File specified could not be opened. Check the permissions and/or the name of the Access File.

(ACC20206) **ACCESS FILE %1 HAS MULTIPLE ENTRIES FOR MASTER %2**

No Access File may have two entries for a single master. Remove all duplicate entries from the Access File

(ACC20207) **NO ENTRY FOUND FOR MASTER %1 IN ACCESS FILE %2**

An Access File specified in the Master must have a corresponding MASTER name. Add a MASTER entry to the Access File, or remove the Access File from the Master.

(ACC20208) **UNABLE TO PARSE ACCESS FILE**

An error occurred while trying to parse the Access File.

(ACC20209) **PARSER ERROR: %1**

There is a grammatical and/or syntactical error in the Access File. If nn/ii/ll is shown then look at line nn, column ii, for a length of ll for the problem token.

(ACC20211) **UNABLE TO PARSE WHERE CLAUSE**

The WHERE clause in the Access File could not be parsed. Check the clause and parse the Master again.

(ACC20212) **ACCESS FILE LOGICAL NAME LENGTH EXCEEDED : %1**

Logical Names: Master File Name, External Location Names, and External Index Names Logical names may not exceed 8 characters.

(ACC20215) **ACCESS FILE SYSTEM ERROR FOR FILE : %1**

Check the physical file name in the Access File.

(ACC20216) **TOO MANY PARTITIONS. PARTITION %1 FOR MASTER %2 EXCEEDS LIMIT.**

There is a limit to the total number of physical partitions that may be used in a request. Delete partitions that are not needed from the Access File.

(ACC20217) **ACCESS FILE TOKEN %1 TOO LONG**

The Token in the Access File is too long. No Token may be longer than 80 characters.

(ACC20219) **PARTITION %2 IS MISSING AN ENTRY FOR LOCATION %1**

If a location entry exists for a horizontal partition, it must be specified for all partitions.

(ACC20224) **WHERE CLAUSE EXCLUDES ALL AVAILABLE PARTITIONS**

The WHERE/IF clause in the current request does not match any of the WHERE clauses specified in the Access File. No partitions can be accessed.

(ACC20225) **UNABLE TO LOAD ACCFILE MODULE**

The system is unable to find a required library.

(ACC20226) **DATASET SPECIFIED FOR MASTER %1 DOES NOT EXIST**

The dataset specified in the Master either has not been created or cannot be read. Verify the status of the specified physical file and try the request again.

(ACC20227) **PARTITION No. %2 FOR DATABASE %1 DOES NOT EXIST**

The partition specified in the Access File for the above Master either has not been created or cannot be read. Verify the status of the physical partition.

(ACC20228) **LOCATION %3 IN PARTITION No. %2 FOR DATABASE %1 DOES NOT EXIST**

The physical file specified for the above LOCATION for the specified
partition in the given database either does not exist or cannot be read. Verify the status of the file and retry the request.

(ACC20235) **INVALID PHYSICAL FILENAME %2 FOR MASTER %1**

Only absolute filenames are accepted by the Access File and the DATASET command.

NF778: Dialogue Manager TRUNCATE Function

The Dialogue Manager TRUNCATE function removes trailing blanks from Dialogue Manager amper variables and adjusts the length accordingly.

Using the Dialogue Manager TRUNCATE Function

The Dialogue Manager TRUNCATE function has only one argument, the string or variable to be truncated. If you attempt to use the Dialogue Manager TRUNCATE function with more than one argument, the following error message is generated:

```
(FOC03665) Error loading external function 'TRUNCATE'
```

This function can only be used in Dialogue Manager commands that support subroutine calls, such as -SET and -IF commands. It cannot be used in -TYPE or -CRTFORM commands or in arguments passed to stored procedures.

Note: A user-written subroutine of the same name can exist without conflict.

Syntax How to Use the TRUNCATE Function

```
-SET &var2 = TRUNCATE(&var1);
```

where:

&var2

Is the Dialogue Manager variable to which the truncated string is returned. The length of this variable is the length of the original string or variable minus the trailing blanks. However, if the original string consisted of only blanks, a single blank, with a length of one is returned.

&var1

Is a Dialogue Manager variable or a literal string enclosed in single quotation marks. System variables and statistical variables are allowed as well as user-created local and global variables.

Example Using the Dialogue Manager TRUNCATE Function

The following example shows the result of truncating trailing blanks:

```
-SET &LONG = 'ABC    ' ;
-SET &RESULT =  TRUNCATE(&LONG);
-SET &LL = &LONG.LENGTH;
-SET &RL = &RESULT.LENGTH;
-TYPE LONG   =  &LONG  LENGTH =  &LL
-TYPE RESULT =  &RESULT LENGTH =  &RL
```

The output is:

```
LONG   =  ABC     LENGTH =  06
RESULT =  ABC LENGTH =  03
```

The following example shows the result of truncating a string of all blanks:

```
-SET &LONG = '       ' ;
-SET &RESULT =  TRUNCATE(&LONG);
-SET &LL = &LONG.LENGTH;
-SET &RL = &RESULT.LENGTH;
-TYPE LONG   =  &LONG  LENGTH =  &LL
-TYPE RESULT =  &RESULT LENGTH =  &RL
```

The output is:

```
LONG   =          LENGTH =  06
RESULT =    LENGTH =  01
```

The following example uses the TRUNCATE function as an argument for EDIT:

```
-SET &LONG = 'ABC    ' ;
-SET &RESULT =  EDIT(TRUNCATE(&LONG)|'Z','9999');
-SET &LL = &LONG.LENGTH;
-SET &RL = &RESULT.LENGTH;
-TYPE LONG   =  &LONG  LENGTH =  &LL
-TYPE RESULT =  &RESULT LENGTH =  &RL
```

The output is:

```
LONG   =  ABC     LENGTH =  06
RESULT =  ABCZ LENGTH =  04
```

NF781: Embedding Text Fields in Headings

You can now embed text fields (FORMAT=TXn) in heading types (heading, footing, subhead, subfoot, and ON TABLE subhead and subfoot) in reports.

Example Embedding a Text Field in a Heading

The following example illustrates how to embed a text field in a heading.

```
TABLE FILE TXTFLD
BY COUNTRY SUBHEAD
"Here is a TX field: <TEXTFLD"
END
```

The heading in the output is:

```
Here is a TX field: The quick brown
                    fox jumps over
                    the lazy dog.
```

The following is the Master File for this example:

```
FILENAME=TXTFLD,SUFFIX=FOC
SEGNAME=ORIGIN,SEGTYPE=S1
FIELDNAME=COUNTRY, FORMAT=A16, $
FIELDNAME=TEXTFLD, FORMAT=TX16, $
```

As is shown in this example, the text field values display on one or more lines. The output is aligned vertically so that the position of the TX field in the initial line is maintained in the following lines.

The number of characters specified in the TX format specification determines the horizontal space occupied by the text field (for example, TX16 means 16 spaces wide).

Reference Usage Notes for Embedding Text Fields in Headings

- You cannot embed TX fields in FML free-text lines.
- HEADING or FOOTING lines can contain multiple embedded TX fields; SUBHEAD or SUBFOOT lines may only contain a single embedded TX field instance.

NF782: Oracle Data Adapter IXSPACE Setting

The new IXSPACE setting for the Oracle Data Adapter enables you to override the default parameters for the Oracle index space implicitly created by the CREATE FILE and HOLD FORMAT SQLORA commands.

Specifying Oracle Index Space Parameters

You can specify up to 94 bytes of index space parameters for the SQL CREATE INDEX statement that the data adapter issues when it creates an Oracle table as a result of a CREATE FILE or HOLD FORMAT SQLORA command for a table that has an index defined.

Syntax How to Specify Oracle Index Space Parameters

You can issue the following SET command at the command prompt, in a stored procedure, or in any supported profile

```
SQL [SQLORA] SET IXSPACE [index_spec]
```

where:

index_spec

Consists of up to 94 bytes of valid Oracle index space parameters. To reset the index space parameters to their default values, issue the SET IXSPACE command with no parameters. If the index specification requires more than one line, use the long form of the SQL SET command:

```
SQL [SQLORA]
SET IXSPACE
index_spec_line1
index_spec_line2
END
```

`SQLORA`

Indicates the Oracle RDBMS. Omit if you issued the SET SQLENGINE command for Oracle.

Examples of index specification parameters you may want to specify are the tablespace clause, global_index clause, local_index clause, and parallel clause. The data adapter does not verify the validity of the parameters. See the *Oracle SQL Reference Manual* for syntax descriptions.

Example Specifying a Table Space for an Oracle Index

When you issue the CREATE FILE or HOLD FORMAT SQLORA command, the DBSPACE setting determines which table space will be used to contain the resulting Oracle table. In prior releases, the index was always placed in the default table space specified at installation.

You can now specify a table space for the index using the SQL SET IXSPACE command. The following example places the index in the tablespace ORATS1:

```
SQL SQLORA SET IXSPACE TABLESPACE ORATS1
```

As a result of setting this value, any CREATE FILE or HOLD FORMAT SQLORA command will generate an SQL CREATE INDEX statement containing the TABLESPACE clause, if an index is defined for the table. You can see the SQL CREATE INDEX syntax in the FSTRACE4 output.

The data adapter does not verify the validity of the parameters. If they are not valid, the following errors are returned at runtime:

```
(FOC1400)    SQLCODE IS 959 (HEX: 000003BF)
             : ORA-00959: tablespace 'tablespacename' does not exist

(FOC1414)    EXECUTE IMMEDIATE ERROR.
```

When the SQL SQLORA SET IXSPACE TABLESPACE *tablespacename* command has been issued with a valid table space name, the following line is added to the SET SQLORA? query command output:

```
(FOCxxxx) DEFAULT IXSPACE IS                    -  : tablespacename
```

If no table space has been set, this line is not displayed.

Example Resetting the IXSPACE Value

The following command resets the IXSPACE parameters to their installation defaults:

```
SQL SQLORA SET IXSPACE
```

NF785: The Adabas Write Data Adapter for FOCUS

You can now use FOCUS to maintain Adabas data sources. This bulletin identifies aspects of the FOCUS file maintenance facilities, MODIFY and Maintain, that are unique for the Adabas environment. MODIFY and Maintain requests read, add, update, and delete records. You can modify single data sources, sets of data sources defined in a multi-segment Master File, or unrelated sets of data sources.

Maintain provides a graphical user interface and event-driven processing. In a Maintain procedure, temporary storage areas called stacks collect data, transaction values, and temporary field values. You can use the Maintain Window Painter facility to design Winforms—windows that display stack values, collect transaction values, and invoke triggers. A trigger implements event-driven processing by associating an action (such as performing a specific case in the Maintain procedure) with an event (such as pressing a particular PF key). Maintain also provides set-based processing through enhanced NEXT, UPDATE, DELETE, and INCLUDE commands.

This bulletin describes differences between the MODIFY and Maintain facilities when these differences affect data adapter processing.

The FOCUS for S/390 documentation contains a detailed discussion of file maintenance with the FOCUS MODIFY and Maintain facilities. Read this documentation carefully before developing MODIFY or Maintain procedures to use with Adabas data sources.

Note: You can maintain up to 64 data sources in a single MODIFY or Maintain procedure. The limit for a MODIFY COMBINE or a Maintain procedure is 16 Master Files; however, each Master File can describe more than one data source, for a total of 64 segments per procedure minus one for the virtual root segment created by the COMBINE command. In addition, a Maintain procedure can call other Maintain procedures that reference additional data sources.

Prerequisites for running MODIFY and Maintain requests include:

- The Write data adapter. The Write component of the data adapter must be installed and operational.
- Proper authorization to perform maintenance operations.
- Existing data sources to modify. If you intend to load new data sources with MODIFY or Maintain, you must generate them first using Adabas utilities.

Activating the Adabas Write Data Adapter

Before you can use any commands that write to an Adabas data source, you must make the following edits to the Master and Access Files:

In the Master File:

- All segments must have SEGTYPE = S0. (For the Read Data Adapter SEGTYPE=S is sufficient.)
- Fields with ACTUAL format Z (zoned) must have a numeric USAGE format (P or I). Format A is supported only for reading an Adabas data source.

  Tip: For best performance, you can change ACTUAL format Z to ACTUAL format P in the Master File. In this case, you must also change the ALIAS attribute to contain the length and data type. If the ALIAS was *ff*, and the field length is *lll*, the ALIAS attribute should be coded as:

  ```
  ALIAS = 'ff,lll,P'
  ```

For example, consider the following field declaration:

```
FIELD = LEAVE_DUE  ,ALIAS = AU       ,USAGE = P2  ,ACTUAL = Z2  ,$
```

You would edit this declaration as follows:

```
FIELD = LEAVE_DUE  ,ALIAS = 'AU,2,P' ,USAGE = P2  ,ACTUAL = P2  ,$
```

- If two or more fields in the Master File are synonyms (they refer to the same field in the data source and, therefore, have the same ALIAS attribute), only the first field encountered in the Master File can be used in INSERT and UPDATE commands. If a synonym other than the first is used in an INSERT command, it will be ignored. If one is used in an UPDATE command, processing will be terminated with the following error message:

  (FOC4565) IGNORED ATTEMPT TO CHANGE NONUPDATABLE FIELD

In the Access File:

- If a segment has the attribute ACCESS=ADBS you can define a unique key by adding the following attribute:

  UNQKEYNAME=*name*

  where:

  name

  Is the name of the elementary or group field to be used as the unique key.

The UNQKEYNAME attribute does not necessarily define the key described in the Adabas FDT (option UQ). The data adapter uses it to decide which rules to apply in a NEXT, INCLUDE, DELETE, or UPDATE command. If the UNQKEYNAME attribute does not correspond to the key described with option UQ in the Adabas FDT, Adabas and the data adapter may not agree on whether a segment instance is unique. This can affect the results of INCLUDE commands, as described in *Effect of UNQKEYNAME on INCLUDE Actions for Segments With ACCESS=ADBS*.

If this attribute is not present, the data adapter uses the rules for modifying a segment with a non-unique key. If it is present, the data adapter uses the rules for modifying a segment with a unique key. Subsequent sections describe these rules.

Note: For modifying Adabas data sources with Maintain, the UNQKEYNAME attribute in the Access File is required. Otherwise you can use only commands that do not change the data source (such as TYPE) and can only issue requests against single-segment data sources.

- We recommend the use of CALLTYPE=FIND instead of CALLTYPE=RL.

Reference AUTOADBS Considerations

You can use AUTOADBS to create a read version of the Master File. To convert it to a Write version, change the SEGTYPE value in all segments to S0.

Limitations on Options Described for the Adabas Data Adapter

When using the Write data adapter, the data adapter automatically sets the values of the following two options:

- ADABAS OPEN is set to YES.
- FETCH is set to OFF for all segments.

Fields That Cannot be Updated

For both MODIFY and Maintain, certain types of fields listed in the Master File cannot be updated. In some cases, the update commands are ignored, and in some cases they generate errors. The following sections contain a detailed explanation.

Reference Using Synonyms

In a Master File, two or more field declarations can refer to the same Adabas field. Each duplicate field declaration after the first is called a synonym. Synonyms can be used in commands that print, but cannot be used in commands that change the data source. The following actions occur as a result of using synonyms in INCLUDE/UPDATE commands:

- Synonym fields used in INCLUDE commands are ignored.
- Synonym fields used in UPDATE commands cause processing to terminate and generate the following error message:

(FOC4565) IGNORED ATTEMPT TO CHANGE NONUPDATABLE FIELD

An UPDATE command was used against a nonupdatable field, i.e. - field described in AFD as SUB/SUPERDESCRIPTOR or part of it;
- field that refers in MFD to the same DB field (ALIAS) as another field (synonym case).

Reference Fields That Cannot Be Updated in MODIFY

In MODIFY, update commands for the following types of fields are ignored:

- Counter fields (ALIAS=*xx*C).
- ORDER fields (ALIAS=ORDER).
- Group fields (UPDATE ignored).

In MODIFY, update commands for the following types of fields generate an error:

- Synonyms (see *Using Synonyms*).
- Fields created from sub- or superdescriptors (as defined in the Access File).

Reference Fields That Cannot Be Updated in Maintain

Maintain generates an error in response to update commands for the following types of fields:

- Counter fields (ALIAS=*xx*C).
- ORDER fields (ALIAS=ORDER).
- Unique key fields (fields specified by the UNQKEYNAME attribute in the Access File).
- Synonyms (see *Using Synonyms*).

Checking Adabas Return Codes and FOCUS Error Message Numbers

The Dialogue Manager status return variable, &RETCODE, indicates the status of FOCUS requests. You can use it to test Adabas return codes. The &RETCODE variable contains the last return code resulting from an executed report request or MODIFY request.

In a Dialogue Manager request, you can use a -IF statement to test the &RETCODE value against a specified Adabas return code. Then, you can take corrective actions based on the result of the -IF test. An Adabas return code of zero (0) indicates a successful execution, a non-zero return code indicates an error.

Another useful Dialogue Manager variable, &FOCERRNUM, stores the last FOCUS or data adapter (not Adabas) error number generated by the execution of a FOCEXEC. See your FOCUS for S/390 documentation for information about &FOCERRNUM and other statistical variables.

Adabas Write Examples

The examples in this document access the EMPLOYEES-WRITE data source.

Adabas Structure of EMPWRITE View
of EMPLOYEES-WRITE Data Source

Example Sample Adabas Write Master File

This Master File provides write access to the Adabas EMPLOYEES-WRITE data source:

```
FILENAME=EMPWRITE,SUFFIX=ADBSIN,$
SEGNAME=S01      ,SEGTYPE=S0,$
 FIELD= EMPLOYEE_ID            ,ALIAS= AA         ,A8    ,A8  ,  INDEX=I,$
 GROUP= FULL_NAME              ,ALIAS= AB         ,A60   ,A60 ,$
  FIELD=FIRST_NAME             ,ALIAS= AC         ,A20   ,A20 ,$
  FIELD=LAST_NAME              ,ALIAS= AE         ,A20   ,A20 ,  INDEX=I,$
  FIELD=MIDDLE_NAME            ,ALIAS= AD         ,A20   ,A20 ,$
  FIELD=ADDRESS_LINE_CNT       ,ALIAS= AIC        ,I4    ,I1  ,$
  FIELD=CITY                   ,ALIAS= AJ         ,A20   ,A20 ,  INDEX=I,$
  FIELD=ZIP_CODE               ,ALIAS= AK         ,A10   ,A10 ,$
  FIELD=POST_CODE              ,ALIAS= AK         ,A10   ,A10 ,$
  FIELD=COUNTRY                ,ALIAS= AL         ,A3    ,A3  ,$
 FIELD= DEPT                   ,ALIAS= AO         ,A6    ,A6  ,  INDEX=I,$
 FIELD= INCOME_CNT             ,ALIAS= AQC        ,I4    ,I1  ,$
 GROUP= LEAVE_DATA             ,ALIAS= A3         ,A16   ,A4  ,$
  FIELD=LEAVE_DUE              ,ALIAS= AU         ,P2    ,Z2  ,$
  FIELD=LEAVE_TAKEN            ,ALIAS= AV         ,P2    ,Z2  ,$
 FIELD= LEAVE_BOOKED_CNT       ,ALIAS= AWC        ,I4    ,I1  ,$
 FIELD= DEPARTMENT             ,ALIAS= S1         ,A4    ,A4  ,  INDEX=I,$
 GROUP= DEPT_PERSON            ,ALIAS= S2         ,A26   ,A26 ,  INDEX=I,$
  FIELD=DEPT_S03               ,ALIAS= AO         ,A6    ,A6  ,  INDEX=I,$
  FIELD=NAME_S03               ,ALIAS= AE         ,A20   ,A20 ,  INDEX=I,$
SEGNAME=AI0101  ,SEGTYPE=S0,PARENT=S01   ,OCCURS=AIC,$ MAX=  8
 FIELD= ADDRESS_LINE           ,ALIAS= AI         ,A20   ,A20 ,$
 FIELD= AI0101_OCC             ,ALIAS= ORDER      ,I4    ,I1  ,$
```

```
SEGNAME=AQ0201  ,SEGTYPE=S0,PARENT=S01   ,OCCURS=AQC,$ MAX= 40
$PEMU = INCOME                ,ALIAS= AQ        ,A19   ,A13 ,$
  FIELD=CURR_CODE             ,ALIAS= AR        ,A3    ,A3  ,$
  FIELD=SALARY                ,ALIAS= AS        ,P9    ,P5  ,$
  FIELD=BONUS_CNT             ,ALIAS= ATC       ,I4    ,I1  ,$
 FIELD= AQ0201_OCC            ,ALIAS= ORDER     ,I4    ,I1  ,$

SEGNAME=AT0301  ,SEGTYPE=S0,PARENT=AQ0201,OCCURS=ATC,$ MAX= 12
 FIELD= BONUS                 ,ALIAS= AT        ,P9    ,P5  ,$
 FIELD= AT0301_OCC            ,ALIAS= ORDER     ,I4    ,I1  ,$

SEGNAME=AW0401  ,SEGTYPE=S0,PARENT=S01   ,OCCURS=AWC,$ MAX= 20
 GROUP= LEAVE_BOOKED          ,ALIAS= AW        ,A16   ,A12 ,$
  FIELD=LEAVE_START           ,ALIAS= AX        ,P6    ,Z6  ,$
  FIELD=LEAVE_END             ,ALIAS= AY        ,P6    ,Z6  ,$
 FIELD= AW0401_OCC            ,ALIAS= ORDER     ,I4    ,I1  ,$
```

Example Sample Adabas Write Access File

The following Access File corresponds to the Master File illustrated in *Sample Adabas Write Master File*:

```
RELEASE=6.2, OPEN=YES,$

$ ADABAS FILE = EMPLOYEES_WRITE                    DICTIONARY =
SEGNAM=S01   ,ACCESS=ADBS,FILENO=042,DBNO=1,CALLTYPE=FIND,
 UNQKEYNAME=EMPLOYEE_ID                                      ,$
 FIELD= DEPARTMENT                          ,TYPE=NOP       ,$
 FIELD= DEPT_PERSON                         ,TYPE=SPR       ,$
  FIELD=DEPT_S03                            ,TYPE=DSC,NU=NO ,$
  FIELD=NAME_S03                            ,TYPE=DSC,NU=NO ,$
SEGNAM=AI0101,ACCESS=MU  ,FILENO=042,DBNO=1,$ ADDRESS_LINE
SEGNAM=AQ0201,ACCESS=PE  ,FILENO=042,DBNO=1,$ INCOME
SEGNAM=AT0301,ACCESS=MU  ,FILENO=042,DBNO=1,$ BONUS
SEGNAM=AW0401,ACCESS=PE  ,FILENO=042,DBNO=1,$ LEAVE_BOOKED
```

Types of Transaction Processing

You can process incoming transactions by comparing (or matching on) the following types of fields:

- For segments with a unique key:
  - A unique key field or superset of the key (unique key plus any number of non-unique key fields or non-key fields.
  - A non-unique key field or a non-key field that is used as a non-unique key field, as described in *Descriptor Considerations*.

- For segments with a non-unique key:
  - Any field when issuing NEXT or INCLUDE commands in a subrequest (after at least one initial MATCH or NEXT command). Any field can be used as a unique key for these commands.

In MODIFY, a MATCH on a non-unique key in a segment with a unique key may retrieve more than one record instance. MATCH returns only the first instance of this answer set; subsequent sections demonstrate how to use NEXT to retrieve the remaining instances.

In Maintain, MATCH always matches on the unique key and retrieves at most one record instance. To match on a non-unique key in Maintain, you use the NEXT command without a prior MATCH. The Maintain implementation of the NEXT command fetches the entire answer set returned by Adabas directly into a stack. It also includes three optional phrases:

- The FOR phrase determines how many records to retrieve.
- The WHERE phrase defines retrieval criteria.
- The INTO phrase names a stack to receive the returned records.

See the FOCUS for S/390 documentation for complete syntax.

Descriptor Considerations

Descriptors (or inverted lists) enhance the performance of data maintenance routines, especially when created on the data source's unique key. Without a descriptor, Adabas must read the entire data source to locate particular record instances; with a descriptor, Adabas can access record instances directly when given search values for the descriptor fields. A data source can have several associated descriptors. Descriptors created for performance reasons can be unique or non-unique. To use FOCUS referential integrity, create descriptors on lower level segments in the Master File.

If all of the following conditions are true, non-descriptor (non-key) fields can be used as non-unique descriptor fields:

- The Access File defaults to or specifies CALLTYPE=FIND.
- The command SET NDFIND=NO was not issued (by default, NDFIND=YES).
- The Adabas session parameter NONDES is set to YES (the default).

You can define a unique key in a data source. When a key is unique, the concatenated values of the key fields in one record cannot be duplicated in any other record. A unique descriptor is generally defined on a unique key. Once you create it, Adabas automatically prevents the insertion of duplicate key values. Any attempt to insert a duplicate record instance generates an error message.

An example of a unique descriptor on a unique key is the employee ID in the sample EMPWRITE data source. Since no two employees can have the same employee number, the value in the EMPLOYEE_ID field makes each record unique:

```
EMPLOYEE_ID  LAST_NAME            FIRST_NAME
-----------  ---------            ----------
50004300     GUERIN               MICHELE
50004600     VERDIE               BERNARD
50004900     CAOUDAL              ALBERT
50005500     BRAUN                ALEXANDRE
50005800     GUENTER              SIMONE
```

You cannot add another record with EMPLOYEE_ID 50004300 to this data source.

Any segment below the root always has a unique key—the ORDER field. You can also use another field as either a non-unique key field (for example, SALARY) or as a unique key, if you are sure that none of the values are duplicated (for example, LEAVE_START).

Modifying Data

With the FOCUS MODIFY and Maintain facilities, you can add new records to a data source, update field values for specific records, or delete specific records.

The data adapter processes a MODIFY or Maintain transaction with the following steps:

1. FOCUS reads the transaction for incoming data values.
2. The data adapter generates the appropriate calls to Adabas based on the MATCH or NEXT criteria.

3. Adabas either returns an answer set consisting of one or more records that satisfy the request, or determines that the record does not exist.
4. After Adabas returns the answer set and/or return code, the data adapter either
   - Performs the update operation (UPDATE or DELETE) on the returned answer set. With MODIFY, the data adapter processes one record at a time; with Maintain, it can either process one record at a time or a set of records.
   - Creates the new record (INCLUDE). In Maintain, it may create multiple records.
5. Adabas changes the database appropriately.

In MODIFY, you must use the NEXT command to process a multi-record answer set one record at a time. Each NEXT command puts you physically at the next logical record in the answer set created by the most recent MATCH command for this segment. In Maintain, one NEXT command can process a multi-record answer set without a prior MATCH.

The MATCH Command

In response to a MATCH command, the data adapter selects the first record in the data source that meets the MATCH criteria.

The MATCH command compares incoming data with one or more field values and then performs actions that depend on whether or not a record with matching field values exists in the data source.

The syntax of the MATCH command in MODIFY is

```
MATCH  field1 [ field2...fieldn ]
  ON MATCH action_1
  ON NOMATCH action_2
```

where:

fieldn

Are fields in any segment of the Master File. FOCUS compares incoming data values against existing field values. The fields can be any combination of key and/or non-key fields. Specify complete fieldnames; MATCH does not support truncated names.

action_1

Is the operation to perform when a record's values match the incoming data values.

action_2

Is the operation to perform when a record's existing values do not match the incoming data values.

The FOCUS for S/390 documentation discusses these actions in detail.

MATCH processing for multi-segment Master Files is the same as for a multi-segment FOCUS database.

Acceptable actions for MATCH commands fall into eight groups. They are operations that:

- Include, change, or delete records.
- Control MATCH processing, such as rejecting the current transaction.
- Read incoming data fields.
- Perform computations and validations, or type messages to the terminal.

- Control case logic.
- Control multiple-record processing.
- Activate and deactivate fields in MODIFY.
- Permanently store data in the Adabas data source.

Data Adapter MATCH Behavior

In MODIFY requests, there are two major differences in the way MATCH commands function for the data adapter and for native FOCUS:

- With the data adapter, you can change the value of a data source's unique key (subject to Adabas limitations) using the UPDATE command. When modifying a FOCUS data source, you cannot change key field values.
- You can MATCH on *any* field or combination of fields in the record. However, if the full unique key is not included in the MATCH criteria, the data adapter may retrieve more than one record as a result of the MATCH.

  For example, if the unique key is EMPLOYEE_ID and the incoming value for MATCH LAST_NAME is SMITH, the answer set contains all records with last name SMITH.

Note: In Maintain, MATCH functions identically for the data adapter and for native FOCUS.

Example Using the MODIFY MATCH Command

Consider a MODIFY request that maintains the EMPWRITE data source. It prompts for an employee ID and for a new number of leave days; then it processes the incoming data. The annotated request contains the following MATCH commands:

```
MODIFY FILE EMPWRITE
PROMPT EMPLOYEE_ID LEAVE_DUE
1. MATCH EMPLOYEE_ID
2. ON MATCH UPDATE LEAVE_DUE
3. ON NOMATCH REJECT
DATA
```

The incoming transaction contains the following values:

```
EMPLOYEE_ID = 12345678
LEAVE_DUE = 20
```

The request processes as follows:

1. The MATCH command compares the value of the incoming EMPLOYEE_ID, 12345678, to the EMPLOYEE_ID values in the records of the EMPWRITE data source. Since EMPLOYEE_ID is the unique key of this data source, Adabas can return at most one record as a result of this MATCH.
2. If a record exists for EMPLOYEE_ID 12345678, the MATCH command updates the LEAVE_DUE value of that record with the incoming value 20.
3. If no record exists for EMPLOYEE_ID 12345678, the MATCH command rejects the transaction.

In Maintain, you do not have to include an ON NOMATCH command in order to reject a transaction; Maintain automatically rejects a transaction that does not satisfy the MATCH criteria.

The NEXT Command

In MODIFY, the NEXT command provides a flexible means of processing multi-record answer sets by moving the current position in the answer set from one record to the next.

The syntax is

```
NEXT field
    ON NEXT action_1
    ON NONEXT action_2
```

where:

field

Is any field in the current segment. This field does not affect subsequent actions.

action_1

Is the operation to perform when there is a subsequent record in the answer set. May be any of the acceptable actions listed for MATCH in *The MATCH Command*.

action_2

Is the operation to perform when no more records exist in the answer set.

The CALLTYPE parameter in the Access File controls the sort order for NEXT. It determines whether to retrieve records in physical order (CALLTYPE=FIND) or sorted by the unique key (CALLTYPE=RL).

Your choice of MATCH and NEXT command combinations determines the contents of the answer set. Subsequent sections explain these choices in more detail:

- NEXT command without a MATCH command. The data adapter requests the retrieval of all records in the data source in physical order (CALLTYPE=FIND) or sorted by the unique key (CALLTYPE=RL).
- MATCH on the unique key. The MATCH returns the single record that is the starting point for any subsequent NEXT commands, which retrieve the remaining records in the answer set in physical order (CALLTYPE=FIND) or sorted by the unique key (CALLTYPE=RL).

- MATCH on a non-unique key field. Adabas returns a multi-record answer set in which each record satisfies the MATCH criteria. The data adapter requests the retrieval of these records in physical order.

You can also use NEXT commands with multi-segment structures (FOCUS views) to modify or display data in either case logic or non-case logic requests. If your MATCH or NEXT specifies a record from a parent segment in a multi-segment structure, that record becomes the current position in the parent segment.

A subsequent NEXT on a field in a descendant of that segment retrieves the first descendant record in the related segment. In MODIFY:

- Without case logic, you can retrieve all parent records in the segment and only the first descendant record of any specified related segment.
- With case logic, you can retrieve all records for each segment defined in a multi-segment Master File. To do so, first MATCH on the parent. Then, in another case, use NEXT to loop through the related segments (at the lowest level) until there are no more related instances. On NONEXT, return to the parent case for the next parent instance.

  You can trace case logic with the FOCUS trace facility. To invoke the trace facility, include the TRACE command on a separate line after the MODIFY FILE command. You can also use the data adapter trace facilities, described in Technical Memo 7966, *Adabas Interface: Using Traces*.

The following sections illustrate different combinations of MATCH and NEXT command with annotated examples. The MODIFY requests have been kept simple for purposes of illustration; you can create more sophisticated procedures.

Note: In Maintain:

- The syntax of the NEXT command includes optional FOR and WHERE phrases that control the number of records retrieved into a stack. As in MODIFY, the CALLTYPE attribute in the Access File determines whether NEXT returns records in physical order or unique key order.
- NEXT always starts its retrieval at the current database position; it will not retrieve a record it has already passed in its retrieval path unless you use the REPOSITION command to reset the current position.

  Also as in MODIFY, once you MATCH on a parent segment, a subsequent NEXT on a child segment retrieves descendant records within the parent established by the MATCH. However, one NEXT command can retrieve all such child instances, without case logic.
- The UPDATE, DELETE, and INCLUDE commands also incorporate the optional FOR phrase to process multiple records from a stack.

For complete details, see the *Maintaining Databases* manual.

NEXT Processing Without MATCH

If you use a NEXT command without a previous MATCH command in a MODIFY request, Adabas returns an answer set consisting of all records in the data source in physical order (CALLTYPE=FIND) or sorted by the unique key (CALLTYPE=RL). Use the ON NEXT command to view each record in the order determined by the CALLTYPE attribute. In a Maintain request, the FOR and WHERE phrases in the NEXT command determine the number of records retrieved, in the order determined by the CALLTYPE attribute.

Example Using NEXT Without MATCH in MODIFY

In this MODIFY example, the NEXT command retrieves each record in physical order because the Access File contains the attribute CALLTYPE=FIND:

```
MODIFY FILE EMPWRITE
NEXT EMPLOYEE_ID
  ON NEXT TYPE "EMPLOYEE ID: <D.EMPLOYEE_ID LAST NAME: <D.LAST_NAME "
  ON NONEXT GOTO EXIT
DATA
END
```

The TYPE commands display the following on the screen:

```
EMPLOYEE ID: 50005800 LAST NAME: GUENTER
EMPLOYEE ID: 50005500 LAST NAME: BRAUN
EMPLOYEE ID: 50004900 LAST NAME: CAOUDAL
EMPLOYEE ID: 50004600 LAST NAME: VERDIE
EMPLOYEE ID: 50004300 LAST NAME: GUERIN
EMPLOYEE ID: 50004200 LAST NAME: VAUZELLE
EMPLOYEE ID: 50004100 LAST NAME: CHAPUIS
EMPLOYEE ID: 50004000 LAST NAME: MONTASSIER
EMPLOYEE ID: 50003800 LAST NAME: JOUSSELIN
EMPLOYEE ID: 50006900 LAST NAME: BAILLET
   .
   .
   .
```

If the Access File contained the attribute CALLTYPE=RL, the records would be retrieved in order of employee ID number.

Example Using NEXT in Maintain

The following Maintain procedure retrieves the same answer set into a stack named INSTACK and displays the retrieved values on a Winform named WIN1 (consult the *Maintaining Databases* manual for instructions on creating Winforms):

```
MAINTAIN FILE EMPWRITE
INFER EMPLOYEE_ID LAST_NAME INTO INSTACK
FOR ALL NEXT EMPLOYEE_ID INTO INSTACK
WINFORM SHOW WIN1
END
```

The following Winform displays as a result of this procedure:

NEXT without prior MATCH

EMPLOYEE_ID	LAST_NAME
50005800	GUENTER
50005500	BRAUN
50004900	CAOUDAL
50004600	VERDIE
50004300	GUERIN
50004200	VAUZELLE
50004100	CHAPUIS
50004000	MONTASSIER
50003800	JOUSSELIN

Exit

NEXT Processing After MATCH on a Unique Key

In MODIFY, NEXT processing is identical for either MATCH on a full unique key or MATCH on a superset (full unique key plus a non-key field).

When the initial MATCH is successful, Adabas retrieves one record. This establishes the logical position in the data source. The subsequent NEXT command causes Adabas to retrieve all records following the matched record in physical order (CALLTYPE=FIND) or key sequence (CALLTYPE=RL).

Example Using NEXT After MATCH on a Full Unique Key in MODIFY

The following is an example of NEXT processing after a MATCH on a full unique key, the EMPLOYEE_ID field:

```
   MODIFY FILE EMPWRITE
   CRTFORM LINE 1
   " PLEASE ENTER VALID EMPLOYEE ID </1"
1. " EMP: <EMPLOYEE_ID  "
2. MATCH EMPLOYEE_ID
   ON NOMATCH REJECT
3. ON MATCH GOTO GETREST
   CASE GETREST
4. NEXT EMPLOYEE_ID
   ON NEXT CRTFORM LINE 10
   " EMPLOYEE_ID: <D.EMPLOYEE_ID   LAST_NAME: <D.LAST_NAME  "
   ON NEXT GOTO GETREST
5. ON NONEXT GOTO EXIT
   ENDCASE
   DATA
   END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID for the search, 20009000.

2. The MATCH command causes Adabas to search the data source for the entered value. If no such record exists, the transaction is rejected.
3. If the specified value matches a value in the EMPLOYEE_ID field of the data source, the procedure branches to the GETREST case; it contains the NEXT command.
4. The NEXT command retrieves the next record based on physical order, if the Access File contains the attribute CALLTYPE=FIND, or based on the sequence of EMPLOYEE_ID, if the Access File specifies CALLTYPE=RL. If such a record exists, the procedure displays the values of the EMPLOYEE_ID and LAST_NAME fields. It continues to display each record in the order determined by the CALLTYPE attribute of the key field, EMPLOYEE_ID.
5. If there are no more records, the procedure ends.

The output after executing this MODIFY procedure with CALLTYPE=FIND is:

```
PLEASE ENTER VALID EMPLOYEE ID                          (line 1)

EMP: 20009000                                          (line 3)

EMPLOYEE_ID:  50005800   LAST_NAME:  GUENTER           (line 10)

EMPLOYEE_ID:  50005500   LAST_NAME:  BRAUN              (line 10)

EMPLOYEE_ID:  50004900   LAST_NAME:  CAOUDAL            (line 10)

EMPLOYEE_ID:  50004600   LAST_NAME:  VERDIE             (line 10)

EMPLOYEE_ID:  50004300   LAST_NAME:  GUERIN             (line 10)
```

Because of the NEXT command, all employees whose records are physically after 20009000 display one at a time on the screen.

The output after executing this MODIFY procedure with CALLTYPE=RL is:

```
PLEASE ENTER VALID EMPLOYEE ID                                (line 1)

EMP: 20009000                                                  (line 3)

EMPLOYEE_ID:  20009100   LAST_NAME:  JENSON                 (line 10)

EMPLOYEE_ID:  20009200   LAST_NAME:  MEYER                  (line 10)

EMPLOYEE_ID:  20009300   LAST_NAME:  SMITH                  (line 10)

EMPLOYEE_ID:  20009400   LAST_NAME:  OLLE                   (line 10)

EMPLOYEE_ID:  20009500   LAST_NAME:  RAY                    (line 10)
```

Notice that the employee ids after 20009000 are retrieved in key sequence.

Example Using NEXT on a Full Unique Key in Maintain

The following Maintain procedure retrieves the same answer set into a stack named EMPSTACK. Assume that when Maintain displays the Winform called WIN1, the user enters the transaction value, 20009000, into a stack named TRANS and clicks the NextRecs button to invoke the NEXTRECS case:

```
MAINTAIN FILE EMPWRITE
INFER EMPLOYEE_ID LAST_NAME INTO EMPSTACK
WINFORM SHOW WIN1
CASE NEXTRECS
  FOR ALL NEXT EMPLOYEE_ID INTO EMPSTACK WHERE EMPLOYEE_ID GT
TRANS.EMPLOYEE_ID
ENDCASE
END
```

The following Winform displays when 20009000 is entered in the EMPLOYEE_ID field with CALLTYPE=FIND:

```
Enter an  employee id:  20009000

EMPLOYEE_ID   LAST_NAME
50005800      GUENTER
50005500      BRAUN
50004900      CAOUDAL
50004600      VERDIE
50004300      GUERIN
50004200      VAUZELLE
50004100      CHAPUIS

NextRecs                Exit
```

NEXT Processing After MATCH on a Non-Unique Key

In a MODIFY request processed by the data adapter, you do not have to MATCH on the full set of unique key fields. You can match on a non-key field or non-unique key. (Maintain always matches on the full unique key, regardless of which fields you specify in the MATCH command.)

When you MATCH on a non-unique key, multiple records may satisfy the MATCH condition. The MATCH operation retrieves the first record of the answer set, and the NEXT command makes the remaining records in the answer set available to the program in physical order. In this case, the order of retrieval for NEXT is always the physical sequence. MATCH on a non-key field should always be processed using CALLTYPE=FIND to prevent the data adapter from issuing an inefficient Read Physical call. For more information see the *FOCUS for IBM Mainframe ADABAS Interface User's Manual and Installation Guide.*

Example Using MATCH on a Non-Unique Key in MODIFY

This annotated procedure is the same procedure described in *Using NEXT After MATCH on a Full Unique Key in MODIFY*, altered to MATCH on the non-unique key field LAST_NAME. The NEXT operation retrieves the subsequent records from the answer set in physical sequence:

```
   MODIFY FILE EMPWRITE
   CRTFORM LINE 1
   "  PLEASE ENTER A LAST NAME </1 "
1. "  LAST NAME: <LAST_NAME </1"
2. MATCH LAST_NAME
   ON NOMATCH REJECT
3. ON MATCH CRTFORM LINE 5
   " EMPLOYEE_ID: <D.EMPLOYEE_ID   LAST_NAME: <D.LAST_NAME "
4. ON MATCH GOTO GETSAME
   CASE GETSAME
5. NEXT LAST_NAME
   ON NEXT CRTFORM LINE 10
   " EMPLOYEE_ID: <D.EMPLOYEE_ID   LAST_NAME: <D.LAST_NAME "
   ON NEXT GOTO GETSAME
6. ON NONEXT GOTO EXIT
   ENDCASE
   DATA
   END
```

The MODIFY procedure processes as follows:

1. The user enters the last name (LAST_NAME) for the search, SMITH.
2. The MATCH command causes Adabas to search the data source for all records with the value SMITH and return them in physical order. If the value SMITH does not exist, the transaction is rejected.
3. If the incoming value matches a value in the data source, the procedure displays the employee ID and last name. (This is the first record of the answer set.)
4. After displaying the record, the procedure goes to the GETSAME case; it uses NEXT to loop through the remaining records in the answer set.

5. Instead of retrieving the next logical record with a higher key value as in the previous example, the procedure retrieves the next record in the answer set (all records in the answer set have the last name SMITH). If any exist, they display on the screen in physical order.
6. When no more records exist with the value SMITH, the procedure ends.

The output from this MODIFY procedure follows:

```
PLEASE ENTER A LAST NAME

LAST_NAME  smith

EMPLOYEE_ID  40000311 LAST_NAME:  SMITH



EMPLOYEE_ID:  20009300   LAST_NAME:  SMITH
EMPLOYEE_ID:  20014100   LAST_NAME:  SMITH
   .
   .
   .
EMPLOYEE_ID:  30038013   LAST_NAME:  SMITH
```

A line displays on the screen for each employee with the last name SMITH. Employee ID 40000311 is the result of the MATCH operation; employee ID 20009300 is the result of the NEXT operation, employee ID 30038013 is the result of the last NEXT operation. Notice that the records are retrieved in physical sequence, and order is not dependent on the CALLTYPE attribute.

Example Using NEXT on a Non-Unique Key Field in Maintain

The following Maintain procedure retrieves the entire answer set into a stack named EMPSTACK. Assume that when Maintain displays the Winform named WINA, the user enters the transaction value (SMITH) into the first record of a stack named TRANS and clicks the NextRecs button to invoke the NEXTRECS case:

```
MAINTAIN FILE EMPWRITE
INFER EMPLOYEE_ID LAST_NAME INTO EMPSTACK
WINFORM SHOW WINA
CASE NEXTRECS
  FOR ALL NEXT EMPLOYEE_ID INTO EMPSTACK WHERE LAST_NAME EQ
TRANS.LAST_NAME
ENDCASE
END
```

The following Winform displays when the NexRecs button is pressed with SMITH entered in the Last_Name field:

```
Last_Name    : SMITH

EMPLOYEE_ID  LAST_NAME
40000311     SMITH
20009300     SMITH
20014100     SMITH
20015400     SMITH
20018800     SMITH
20023600     SMITH

NextRecs        Exit
```

INCLUDE, UPDATE, and DELETE Processing

While MATCH and NEXT operations in MODIFY can operate on unique key or non-unique key fields and return single or multi-record answer sets, the MODIFY commands INCLUDE, UPDATE, and DELETE must always identify the target records by their unique key. Therefore, in MODIFY, each update operation affects at most one record. (In Maintain, the FOR phrase in the update command determines the number of records affected.)

Reference Rules for Inserting Records Into an Adabas Data Source

- For a segment with a unique key:

  The unique key field value is used to insert the target segment. If any additional MATCH criteria are supplied for a segment in the path, they will be used to qualify that path segment.

  An attempt to use MATCH on a non-unique key before an INCLUDE command generates the following error message:

  ```
  (FOC4563)INCORRECT INCLUDE REQUEST FOR NON UNIQUE KEY
  ```

- For a segment with a non-unique key or no key:

  If you want to insert an additional record for an existing key field, you must MATCH on the key field and specify ON MATCH INCLUDE.

- For segments with ACCESS=PE or MU, if a new occurrence is inserted, you must set the occurrence number (ORDER field) to 0 (zero indicates the next occurrence) or to a value greater than the number of existing occurrences. This new occurrence is always inserted after the last existing occurrence. For example, if a PE or MU segment has two existing occurrences, the next occurrence added will always be the third. If the occurrence with the given number already exists, processing will terminate with the following message:

  ```
  (FOC4564)  THIS OCCURRENCE ALREADY EXISTS. USE UPDATE COMMAND.
  ```

  You should use the UPDATE command instead of INCLUDE in this case.

- For segments with ACCESS=MU, if the corresponding field with the MU option is defined *without* the NU option on the second level of the Adabas FDT (that is, it is part of a periodic group (PE) or simple group (GR)):

  - If an INCLUDE command is issued for the root segment alone, Adabas automatically inserts an empty child (MU) segment and, if this child segment has a parent with ACCESS=PE, an empty parent segment.
  - If an INCLUDE command is issued for the root and a parent (PE) of the MU segment simultaneously, but without values for the child (MU) segment, Adabas automatically inserts an empty child (MU) segment. The data adapter automatically inserts the first occurrence of the parent (PE) segment using the values from the INCLUDE command.
  - If an INCLUDE command is issued for the root, parent (PE), and child (MU) segments simultaneously, the data adapter inserts the first occurrence of both the child and parent segments using the values from the INCLUDE command.
- For segments with ACCESS=MU, if the corresponding field with the MU option is defined *with* the NU option on *any* level in the Adabas FDT, Adabas automatically suppresses the empty values. An INCLUDE command can be issued for the parent and child segments separately or simultaneously.

Note: The Master File does not identify the level of an MU field because if the MU field belongs to a simple group, the Master File does not describe this group. (However, if AUTOADBS created the Master File, there will be a commented declaration for the group.) Therefore, FOCUS cannot determine whether Adabas will add an empty occurrence for the corresponding segment. You must be aware of the options in the Adabas FDT to understand the behavior that occurs in response to an INCLUDE command for an MU segment.

Effect of UNQKEYNAME on INCLUDE Actions for Segments With ACCESS=ADBS

The UNQKEYNAME attribute in the Access File determines how the data adapter presents an INCLUDE command to Adabas. The option UQ in the Adabas FDT and the specific fields used in the MATCH command determine whether Adabas actually inserts the segment instance. The following table describes how these factors affect the result of the INCLUDE command. Assume that the Access File specifies UNQKEYNAME=EMPLOYEE_ID and that the employee id value EMPID005 already exists in the data source:

Result of INCLUDE Command for Existing EMPLOYEE_ID EMPID005

EMPLOYEE_ID has option UQ in FDT	Fields in MATCH command	Instance Inserted (ON NOMATCH)
No	EMPLOYEE_ID only	ON MATCH case performed
Yes	EMPLOYEE_ID only	ON MATCH case performed

EMPLOYEE_ID has option UQ in FDT	Fields in MATCH command	Instance Inserted (ON NOMATCH)
No	EMPLOYEE_ID plus fields with values that do not already exist	Yes
Yes	EMPLOYEE_ID plus fields with values that do not already exist	No - error (FOC4561), RC=198

This table describes INCLUDE actions when EMPLOYEE_ID is not in the MATCH field list:

Result of INCLUDE Command Without Matching on Field EMPLOYEE_ID

UNQKEYNAME = EMPLOYEE_ID?	Instance Inserted
Yes	No - error (FOC4563)
No	Yes

Reference Rules for Deleting Records From an Adabas Data Source

- For a segment with a unique key, the key field value is used to delete the target segment.

- For a segment with a non-unique key, you must supply the key field. If the MATCH criteria for this type of segment, in the path or target, do not identify a unique occurrence, the first occurrence found will be deleted. The use of Adabas descriptors for this type of segment is highly recommended for efficiency.
- For a segment with no key, you must supply at least one MATCH condition. If the MATCH criteria for this type of segment, in the path or target, do not identify a unique occurrence, the first occurrence found will be deleted.
- A segment occurrence with ACCESS=PE or MU is deleted from the data source except if it is the last occurrence for an ACCESS=PE segment. Adabas will only delete the last occurrence if all fields have the NU option in the FDT; if they do not all have this option, the occurrence will have empty values in all fields.
- When you delete segments that have dependent segments, the DELETED counter for the session may have an incorrect value. For a first level segment with descendants, this counter will always be incorrect. For a second level segment, this counter will be incorrect if there are multiple descendant segments. For a third level segment, this counter is always correct.

Reference Rules for Updating Records in an Adabas Data Source

- For a segment with a unique key, the key field value and any additional MATCH criteria are used to qualify the target segment for update.
- For a segment with a non-unique key, you must supply the key field. If the MATCH criteria for this type of segment, in the path or target, do not identify a unique occurrence, the first occurrence found will be updated. The use of Adabas descriptors for this type of segment is highly recommended for efficiency.
- For a segment with no key, you must supply at least one MATCH condition. If the MATCH criteria for this type of segment, in the path or target, do not identify a unique occurrence, the first occurrence found will be updated.

Example Updating Adabas Records With MODIFY

Suppose you want to display all the employees in a department and update the amount of leave they have taken:

```
   MODIFY FILE EMPWRITE
   CRTFORM LINE 1
   " PLEASE ENTER A VALID DEPARTMENT </1"
1. " DEPARTMENT: <DEPARTMENT "
2. MATCH DEPARTMENT
   ON NOMATCH REJECT
   ON MATCH CRTFORM LINE 10
3. "ID: <D.EMPLOYEE_ID LEAVE DUE: <D.LEAVE_DUE> TAKEN <T.LEAVE_TAKEN> "
4. ON MATCH UPDATE LEAVE_TAKEN
   ON MATCH GOTO GETREST
   CASE GETREST
5. NEXT EMPLOYEE_ID
   ON NEXT CRTFORM LINE 10
   "ID: <D.EMPLOYEE_ID LEAVE DUE: <D.LEAVE_DUE> TAKEN <T.LEAVE_TAKEN> "
   ON NEXT UPDATE LEAVE_TAKEN
   ON NEXT GOTO GETREST
6. ON NONEXT GOTO EXIT
   ENDCASE
   DATA
   END
```

The MODIFY procedure processes as follows:

1. The user enters the department (DEPARTMENT) for the search, PROD.
2. The MATCH command causes Adabas to search the data source for the first record with the value PROD and return them in physical sequence. If none exists, the transaction is rejected.
3. If the supplied value matches a database value, the procedure displays it.

4. The procedure updates the LEAVE_TAKEN field for the first retrieved record using the turnaround value from the CRTFORM. EMPLOYEE_ID establishes the target record for the update.
5. Each time it executes the NEXT, the procedure retrieves the next record with the same department, PROD. It displays each one in physical order. It updates the LEAVE_TAKEN field for each retrieved record with the turnaround value.
6. When no more records exist for department PROD, the procedure ends.

Example Updating Adabas Records With Maintain

In Maintain, you can use stack columns as turnaround values to update a data source. The following annotated Maintain request updates the same records as the preceding MODIFY request:

```
   MAINTAIN FILE EMPWRITE
   INFER EMPLOYEE_ID LEAVE_DUE LEAVE_TAKEN INTO EMPSTACK
1. WINFORM SHOW WIN1
2. CASE MATCHREC
     FOR ALL NEXT EMPLOYEE_ID INTO EMPSTACK
    WHERE DEPARTMENT EQ VALSTACK.DEPARTMENT
   ENDCASE
3. CASE UPDLV
    FOR ALL UPDATE LEAVE_TAKEN FROM EMPSTACK
   ENDCASE
   END
```

The Maintain procedure processes as follows:

1. A Winform named WIN1 displays. Assume that it displays an entry field labeled DEPARTMENT (whose source and destination stack is called VALSTACK) and a grid (scrollable data source) with columns EMPLOYEE_ID, LEAVE_DUE and LEAVE_TAKEN. See the *Maintaining Databases* manual for instructions on creating Winforms.
2. The user enters a DEPARTMENT value for the search and clicks the GetEmps button to invoke case MATCHREC. Case MATCHREC retrieves the records that satisfy the NEXT criteria and stores them in a stack named EMPSTACK. The Winform displays the retrieved records on the grid.
3. The user edits all the necessary leaves taken directly on the Winform grid and then clicks the Update button to invoke case UPDLV, which updates all leaves taken.

The following Winform displays when the GetEmp button (or PF4) is pressed with PROD entered in the Department field:

Department : PROD

EMPLOYEE_ID	LEAVE_DUE	LEAVE_TAKEN
11100305	31	12
11100308	31	5
11100309	30	28
11100310	30	5
11100311	30	10
11100312	29	28
11100313	30	0

GetEmp (PF4) Update (PF6) Exit (PF3)

The user can update the LEAVE_TAKEN field for all of the listed employees and update them all in one step with the Update button (or PF6).

Adabas Transaction Control Within MODIFY

The data adapter supports the Logical Unit of Work (LUW) concept. An LUW consists of one or more FOCUS maintenance actions (UPDATE, INCLUDE, or DELETE) that process as a single unit. The maintenance operations within the LUW can operate on the same or separate data sources.

In MODIFY, all records read by MATCH and NEXT commands are held by Adabas in a user's hold record queue. To prevent overflow of this queue, the user should periodically issue the SQL COMMIT WORK command to propagate the changes to the Adabas data source and clear the queue.

In Maintain, records are not held when they are read into a stack. They are held only in response to an updating command.

A transaction is defined as all actions taken since the application first accessed Adabas, last issued an SQL COMMIT WORK in MODIFY or COMMIT command in Maintain, or last issued an SQL ROLLBACK WORK in MODIFY or ROLLBACK command in Maintain.

Within a Logical Unit of Work, Adabas either executes all commands completely, or else it executes none of them. If Adabas detects no errors in any of the commands within the LUW:

- FOCUS issues a COMMIT WORK command. The data adapter issues an ET (End of Transaction) command to Adabas. The changes indicated by the updates within the transaction are recorded in the data source.
- Adabas releases locks on the target data.
- Database changes become available for other tasks.

In response to unsuccessful execution of any command in the transaction, the data adapter:

- Issues a BT (Backout Transaction) command to Adabas. Target data returns to its state prior to the unsuccessful transaction. All changes attempted by the commands in the transaction are backed out.
- Does not execute the remaining commands in the transaction.
- Releases locks on the target data.
- Discards partially accumulated results.

Adabas and the data adapter provide a level of automatic transaction management but, in many cases, this level of management alone is not sufficient. FOCUS supports explicit control of Adabas transactions with the commands SQL COMMIT WORK and SQL ROLLBACK WORK in MODIFY, and with the commands COMMIT and ROLLBACK in Maintain.

Note: SQL COMMIT WORK and SQL ROLLBACK WORK are data adapter commands. Do not confuse these commands with the FOCUS COMMIT WORK and ROLLBACK WORK commands that apply to FOCUS databases only. The data adapter ignores COMMIT WORK and ROLLBACK WORK without the SQL qualifier.

Unless you specify SQL COMMIT WORK and/or SQL ROLLBACK WORK in your MODIFY procedure (or COMMIT and/or ROLLBACK in your Maintain procedure), all FOCUS maintenance actions until the END command constitute a single LUW. If the procedure completes successfully, the data adapter automatically transmits an ET command to Adabas, and the changes become permanent. If the procedure terminates abnormally, the data adapter issues a BT command to Adabas, and the database remains untouched. Since locks are not released until the end of the program, a long MODIFY or Maintain procedure that relies on the default, end-of-program ET can interfere with concurrent access to data. In addition:

- You may lose all updates in the event of a system failure.
- You may fill up the hold queue Adabas establishes records locked by a user. The number of ISNs that can be held in this queue is determined by the Adabas NISNHQ parameter. To avoid exceeding this number, keep a counter in your MODIFY or Maintain procedure, and commit or rollback the transaction to avoid holding too many records.

Transaction Termination (COMMIT WORK)

The SQL COMMIT WORK command signals the successful completion of a transaction at the request of the procedure. Execution of a COMMIT command makes changes to the data sources permanent. The syntax in a MODIFY request is:

```
SQL COMMIT WORK
```

You can issue a COMMIT WORK as an ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT condition, after an update operation (INCLUDE, UPDATE, DELETE), or within cases of a case logic request.

Note: In Maintain, you must use the Maintain facility's COMMIT command to transmit an ET (End of Transaction) command to Adabas.

Example Using COMMIT WORK in a MODIFY Procedure

A COMMIT WORK example using Case Logic follows:

```
CASE PROCESS
   CRTFORM
   MATCH field1 ...
     ON MATCH insert, update, delete, ...
   GOTO EXACT
ENDCASE
CASE EXACT
   SQL COMMIT WORK
   GOTO TOP
ENDCASE
```

The PROCESS case handles the MATCH, ON MATCH, ON NOMATCH processing. Then it transfers to CASE EXACT, which commits the data instructing Adabas to write the entire Logical Unit of Work to the database.

Transaction Termination (ROLLBACK WORK)

The SQL ROLLBACK WORK command signals the unsuccessful completion of a transaction at the request of the procedure. Execution of a ROLLBACK command backs out all changes made to the data sources since the last COMMIT command. The syntax in a MODIFY request is:

```
SQL ROLLBACK WORK
```

You can design a MODIFY procedure to issue a ROLLBACK WORK command if you detect an error. For example, if a FOCUS VALIDATE test finds an inaccurate input value, you may choose to exit the transaction, backing out all changes since the last COMMIT. You can issue ROLLBACK WORK as an ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT condition, or within cases of a case logic request.

Note: In Maintain, you must use the Maintain facility's ROLLBACK command to transmit a BT command (Back out Transaction) to Adabas.

The data adapter automatically executes an SQL ROLLBACK WORK command when you exit from a transaction early. For example, if you exit a CRTFORM without specifying some action, the data adapter automatically issues a ROLLBACK WORK command on your behalf.

The data adapter automatically issues a BT command in case of system failure or when it detects a fatal data error, such as a reference to a field or data source that does not exist.

Example Using ROLLBACK WORK in a MODIFY Procedure

The following is an example of the ROLLBACK WORK command using Case Logic:

```
ON NOMATCH CRTFORM ...
ON NOMATCH VALIDATE ...
    ON INVALID GOTO ROLLCASE
        .
        .
        .
CASE ROLLCASE
    SQL ROLLBACK WORK
    GOTO TOP
ENDCASE
```

Code the ROLLBACK WORK command before a REJECT command. FOCUS ignores any action following the rejection of a transaction, except for GOTO or PERFORM.

For example:

```
ON MATCH SQL ROLLBACK WORK
ON MATCH REJECT
```

Example Transaction Control in Adabas

Each time an employee takes leave days, the following example updates the LEAVE_TAKEN field in the root segment of the EMPWRITE data source and posts a record for the leave start and end dates in the related AW0401 segment. To ensure that both updates complete or neither one does, the MODIFY procedure places both actions prior to a COMMIT WORK command. If the descendant data source is not processed, ROLLBACK WORK discards the whole logical transaction.

```
MODIFY FILE EMPWRITE
CRTFORM LINE 1
"</2 <25 MODIFY FOR LEAVE TAKEN </2 "
"<20 ENTER THE EMPLOYEE ID <EMPLOYEE_ID "
MATCH EMPLOYEE_ID
  ON MATCH CRTFORM LINE 7
  "<D.FIRST_NAME <D.LAST_NAME> LEAVE TAKEN <T.LEAVE_TAKEN> </1 "
  ON MATCH UPDATE LEAVE_TAKEN
  ON MATCH COMPUTE AW0401_OCC =0;
  ON NOMATCH REJECT
MATCH AW0401_OCC
  ON NOMATCH CRTFORM LINE 10
  "PLEASE ENTER LEAVE DATES"
  "LEAVE_START: <T.LEAVE_START> "
  "LEAVE_END:   <T.LEAVE_END> "
  ON NOMATCH INCLUDE
  ON NOMATCH SQL COMMIT WORK
  ON MATCH SQL ROLLBACK WORK
  ON MATCH REJECT
DATA
END
```

Using the Return Code Variable: FOCERROR

FOCUS stores the return code from the updating commands INCLUDE, DELETE, and UPDATE in the variable FOCERROR:

- In MODIFY, the return code value is the Adabas response code.
- In Maintain, the value is
  - 1 if an INCLUDE command failed.
  - 2 if a DELETE command failed.
  - 3 if an UPDATE command failed.

A return code of zero indicates successful completion of the last updating command issued.

You can test the FOCERROR variable and take appropriate action if you encounter a non-fatal error. This condition might indicate the need to ROLLBACK the transaction or re-prompt the user for new input values. In Maintain, all errors after updating commands are non-fatal, and you should always test FOCERROR after issuing an updating command. In MODIFY, you can issue the SQL SET ERRORRUN ON command to make these errors non-fatal. If you do not issue this command, all errors after updating commands in MODIFY will be fatal.

All errors that result from retrieval commands such as MATCH or NEXT are fatal errors. An example of a command that causes a retrieval error is attempting to read a record held by another user. These errors terminate MODIFY and Maintain procedures.

For a list of common Adabas response codes, see *Adabas Response Codes*.

Using the Data Adapter SET ERRORRUN Command

With SET ERRORRUN ON, MODIFY processing continues even when a serious error occurs, allowing applications to handle their own errors in the event that an Adabas error is part of the normal application flow. Code this command explicitly within the MODIFY procedure, preferably in CASE AT START, where it executes once.

Note: Maintain does not support the SET ERRORRUN command. All errors after updating commands are non-fatal.

When SET ERRORRUN is ON, the MODIFY procedure reports the error but continues execution. The MODIFY code can then test the value of FOCERROR to determine the cause of the error and take appropriate action. Be careful in evaluating the contents of FOCERROR, to prevent unpredictable errors in subsequent MODIFY processing.

SET ERRORRUN returns to its default setting of OFF at the end of the MODIFY procedure.

Syntax How to Issue the SET ERRORRUN Command in a MODIFY Procedure

```
CASE AT START
    SQL SET ERRORRUN {OFF|ON}
ENDCASE
```

where:

OFF

Stops MODIFY processing when the Data Adapter detects a fatal error. OFF is the default.

ON

Enables MODIFY processing to continue despite fatal errors. Test the value of FOCERROR to determine the desired action after an updating command fails. After the procedure ends, ERRORRUN returns to its default value of OFF.

Modifying Data sources Without Unique Keys

Adabas permits data sources with duplicate records. Such data sources cannot possibly have a unique key, since no combination of field values can make a given record unique.

The data adapter provides a way of maintaining data sources with duplicate records. However, only the first record encountered will be affected by data maintenance commands.

Note:

- Maintain does not support modifying unkeyed data sources.

Referential Integrity

The term *referential integrity* defines the type of consistency that should exist between parent and descendant segments.

- A parent segment must exist before a related record in a lower level segment can exist. For example, a specific employee ID must exist in the EMPWRITE data source before a salary can be added for that employee in the AQ0201 segment (INCLUDE referential integrity).
- If a parent segment is deleted, all of its descendant segments must be deleted (DELETE referential integrity).

FOCUS can provide referential integrity for the following types of data sources described in one multi-segment Master File:

- A single Adabas file.
- Multiple Adabas files connected by an embedded join in the Access File. The segment declarations for all such Adabas files must have ACCESS=ADBS in the Access File and must be joined using the Access File attributes KEYFLD and IXFLD. For more information on embedded joins, see the *FOCUS for IBM Mainframe ADABAS Interface User's Manual and Installation Guide.*

The following sections discuss referential integrity constraints.

FOCUS INCLUDE Referential Integrity

The FOCUS MODIFY facility syntax provides automatic referential integrity for inserting new records.

You must describe the data source in one multi-segment Master File. The multi-segment description establishes the relationship between the segments.

With a multi-segment Master File, you cannot add a descendant segment using the FOCUS MODIFY facility unless the parent segment already exists. Therefore, a MODIFY procedure that inserts records must MATCH on the parent segment before adding a record in descendant segment.

Example Using FOCUS INCLUDE Referential Integrity

The following examples demonstrate referential integrity when adding new records. The scenarios are:

1. Add a salary for an employee *only* if data for the employee ID already exists.
2. The employee ID does not exist. Add both a new employee ID and a salary.

A simple, annotated FOCUS MODIFY procedure for each scenario follows.

The first example adds course information only if a record already exists for the employee:

```
   MODIFY FILE EMPWRITE
   CRTFORM LINE 2
   "ADD SALARY INFORMATION FOR EMPLOYEE  </1"
1. "EMPLOYEE ID: <EMPLOYEE_ID  </1 "
   "SALARY: <SALARY          CURR_CODE: <CURR_CODE "
2. MATCH EMPLOYEE_ID
   ON MATCH COMPUTE AQ0201_OCC = 0;
3. ON MATCH CONTINUE
4. ON NOMATCH REJECT
5. MATCH AQ0201_OCC
       ON NOMATCH INCLUDE
       ON MATCH REJECT
   DATA
   END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID, salary, and appropriate code. This constitutes the incoming transaction record.
2. The MATCH command causes Adabas to search the data source for an existing record with the specified employee ID.
3. If the employee record exists, the MODIFY sets the salary occurrence number to zero and continues to the next MATCH command.
4. If no record in the EMPWRITE data source exists with the specified employee ID, MODIFY rejects this transaction and routes control to the top of the procedure.

5. MATCH AQ0201_OCC causes Adabas to search for an existing salary record with the specified occurrence number for the employee ID located in Step 2. Because the occurrence number was set to zero in the previous MATCH command, no such record exists, and the MODIFY adds a new salary occurrence. If the occurrence number already existed, the MODIFY would reject the transaction as a duplicate.

The second example adds a record to the EMPWRITE data source for a new employee and adds a salary for that employee to the AQ0201 segment. If the employee ID already exists, the procedure adds only the salary information to the AQ0201 segment:

```
   MODIFY FILE EMPWRITE
   CRTFORM LINE 1
1. "ID: <EMPLOYEE_ID "
2. MATCH EMPLOYEE_ID
3. ON NOMATCH CRTFORM LINE 2
   "       LAST: <LAST_NAME        FIRST: <FIRST_NAME </1 "
   " MIDDLE: <MIDDLE_NAME> </1 "
   "  ADDRESS_LINE_CNT <ADDRESS_LINE_CNT> CITY:  <CITY "
   "  ZIP: <ZIP_CODE    COUNTRY: <COUNTRY  </1"
   " DEPT: <DEPT   INCOME_CNT: <INCOME_CNT  </1"
   "LEAVE DUE: <LEAVE_DUE      DEPARTMENT: <DEPARTMENT "
   "SALARY: <SALARY  CODE: <CURR_CODE "
   ON NOMATCH INCLUDE
   ON NOMATCH COMPUTE AQ0201_OCC = 0;
   ON MATCH COMPUTE AQ0201_OCC = 0;
4. ON MATCH CRTFORM LINE 9
   "SALARY: <SALARY  CODE: <CURR_CODE "R_TAKEN  QTR: <QTR "
```

5.
```
MATCH AQ0201_OCC
ON NOMATCH INCLUDE
ON MATCH REJECT
DATA
END
```

The MODIFY procedure processes as follows:

1. The user enters EMPLOYEE_ID.
2. The MATCH command causes Adabas to search the EMPWRITE data source for an existing record for the specified employee ID.
3. If the employee record does not exist, the user enters the data for both the employee and the salary. The procedure adds a record to each segment.
4. If the employee already exists, the user enters only the salary data.
5. The MATCH AQ0201_OCC command causes Adabas to search the AQ0201 segment for the specified occurrence. If this occurrence does not exist for this employee, the procedure adds it. If it does exist, the procedure rejects the transaction.

FOCUS DELETE Referential Integrity

FOCUS provides automatic referential integrity for deleting records described in a multi-segment Master File. Just as with INCLUDE referential integrity, only data sources described in a multi-segment Master File invoke FOCUS DELETE referential integrity.

When you delete a parent segment in a MODIFY or Maintain procedure, FOCUS automatically deletes all descendant segments at the same time.

Example Using FOCUS DELETE Referential Integrity

For example, when you delete an employee from the root segment in the EMPWRITE Master File, FOCUS also deletes all records from the descendent segments for the employee:

```
MODIFY FILE EMPWRITE
CRTFORM LINE 2
"DELETE EMPLOYEE </1"
```

1. `"EMPLOYEE ID: <EMPLOYEE_ID   "`
2.
```
MATCH EMPLOYEE_ID
ON MATCH COMPUTE DOIT/A1 = 'N';
ON MATCH CRTFORM LINE 6
```
3.
```
"EMPLOYEE TO BE DELETED: <D.EMPLOYEE_ID </1"
"          LAST NAME:   <D.LAST_NAME </1"
"          FIRST NAME:  <D.FIRST_NAME </1"
"          DEPARTMENT:  <D.DEPARTMENT </1"
"IS THIS THE EMPLOYEE YOU WISH TO DELETE? (Y,N): <DOIT "
ON MATCH IF DOIT EQ 'N' THEN GOTO TOP;
```
4.
```
ON MATCH DELETE
ON NOMATCH REJECT
DATA
END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID.
2. The MATCH command causes Adabas to search the root segment of the EMPWRITE data source for an existing record with the specified employee ID.
3. If the record exists, the MODIFY displays information for verification.

4. Once verified, FOCUS deletes the employee and all associated segment instances. When FOCUS deletes a parent instance, it automatically deletes all associated related instances.

Inhibiting FOCUS Referential Integrity

You may not always want to enforce FOCUS referential integrity. If the referential integrity is being enforced between separate Adabas files described in a single Master File using an embedded join described in the Access File, you can describe each file you want to modify separately in a separate Master File.

Another technique is to COMBINE data sources rather than using a multi-segment Master File. COMBINE of single data sources does not invoke FOCUS referential integrity.

The MODIFY COMBINE Facility

Some applications require that you use a single input transaction to update several data sources in the same MODIFY procedure. If the data sources are not defined in the same Master File, you can use the COMBINE facility to modify them as if they are one.

Note: In Maintain, you do not issue a COMBINE command to modify unrelated data sources. Instead, you reference multiple data sources in the MAINTAIN FILE command. For example:

```
MAINTAIN FILES EMPWRITE AND COURSE
```

You can maintain up to 63 data sources in a single MODIFY procedure that operates on a COMBINE structure. The COMBINE limit is 16 Master Files; however, each Master File can describe more than one data source, for a total of 64 per procedure, minus one for the virtual root segment created by the COMBINE command.

The COMBINE facility links multiple data sources and assigns a new name to them so FOCUS can treat the data sources as a single structure. Data sources in a COMBINE structure can have different SUFFIX attributes, but you cannot combine a FOCUS database with anything except other FOCUS databases.

Note: In Maintain, you can modify FOCUS databases and Adabas data sources in the same procedure.

When you issue a COMBINE command, the COMBINE structure remains in effect for the duration of the FOCUS session or until you enter another COMBINE command. Only one COMBINE structure can exist at a time, so each subsequent COMBINE command replaces the existing structure.

Do not confuse COMBINE with the dynamic JOIN command. You use JOIN to *report* from multiple data sources that share at least one common field or for LOOKUP functions. With the COMBINE facility, you can *MODIFY* multiple data sources. COMBINE is part of the MODIFY command; only the MODIFY and CHECK FILE commands process COMBINE structures. The FIND function also works in conjunction with COMBINE (see *The FIND Function*).

Note that COMBINE considers the component structures to be unrelated. FOCUS referential integrity does not apply to a COMBINE of single-data source Master Files. Your procedure should check for and enforce referential integrity, if necessary.

Syntax How to Use the COMBINE Command

The basic syntax for the COMBINE command is

```
COMBINE FILES  file1 [PREFIX pref1|TAG tag1] [AND]
    .
    .
    .
               filen [PREFIX prefn|TAG tagn] AS asname
```

where:

file1 - filen

Are the Master File names of the data sources you want to modify. You can specify up to 16 Master Files.

pref1 - prefn

Are prefix strings for each file; up to four characters. They provide uniqueness for fieldnames. You cannot mix TAG and PREFIX in a COMBINE structure. Refer to the FOCUS for S/390 documentation for additional information.

tag1 - tagn

Are aliases for the data source names; up to eight characters. FOCUS uses the tag name as a qualifier for fields that refer to that data source in the combined structure. You cannot mix TAG and PREFIX in a COMBINE, and you can only use TAG if FIELDNAME is set to NEW or NOTRUNC.

AND

Is an optional word to enhance readability.

asname

Is the required name of the combined structure to use in MODIFY procedures and CHECK FILE commands.

Once you enter the COMBINE command, you can modify the combined structure.

How FOCUS Creates a COMBINE Structure

For example, the EMPINFO data source contains employee number, last name, first name, hire date, department code, current job code, current salary, number of education hours, and bonus plan information. A second data source, PAYINFO, is a historical record of the employee's pay history. It contains the employee number, date of increase, percent of increase, new salary, and job code.

Each time a salary changes, both the EMPINFO and PAYINFO data sources must reflect the change. Since both data sources need to share data entered for employee number, salary and job code, this application is appropriate for the COMBINE facility. You can update both data sources at the same time without having to define multi-segment Master and Access Files.

The following figures represent the data sources as separate entities.

```
EMPINFO data source          PAYINFO data source

           EMPINFO                  PAYINFO
01         S0          01           S0
**************         **************
*EMPLOYEE_ID **        *PAYEID       **
*LAST_NAME   **        *DAT_INC      **
*FIRST_NAME  **        *PCT_INC      **
*HIRE_DATE   **        *SALARY       **
*            **        *             **
***************        ***************
**************         **************
          EMPINFO                 PAYINFO
```

To modify the data sources simultaneously, issue the following sequence of commands at the FOCUS command level or in a FOCEXEC:

```
COMBINE FILES EMPINFO PAYINFO AS EMPSPAY
MODIFY FILE EMPSPAY
    .
    .
    .
```

In the following picture, generated by the CHECK FILE command, FOCUS defines a new segment, identified as SYSTEM99, to be the root segment of the combined structure.

SYSTEM99 acts as the traffic controller for this structure; it is a virtual (artificial) segment. It counts as one segment towards the total of 64 segments allowed in the COMBINE structure.

```
check file empspay pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   3  ( REAL=    3  VIRTUAL=   0 )
  NUMBER OF FIELDS=    15  INDEXES=   0  FILES=     3
  TOTAL LENGTH OF ALL FIELDS=   95
 SECTION 01
              STRUCTURE OF SQLDS    FILE EMPINFO  ON 07/22/93 AT 09.54.27

          SYSTEM99
  01      S0
 **************
 *            **
 *            **
 *            **
 *            **
 *            **
 ***************
  **************
        I
        +-----------------+
        I                 I
        I EMPINFO         I PAYINFO
  02    I S0        03    I S0
 **************    **************
 *EMPLOYEE_ID **   *PAYEID      **
 *LAST_NAME   **   *DAT_INC     **
 *FIRST_NAME  **   *PCT_INC     **
 *HIRE_DATE   **   *SALARY      **
 *            **   *            **
 ***************   ***************
  **************    **************
          EMPINFO           PAYINFO
```

The COMBINE facility makes it easy to modify many files with the same transaction. For additional information regarding the COMBINE facility of MODIFY, refer to the *Maintaining Databases* manual.

The LOOKUP Function

The LOOKUP function, used in FOCUS MODIFY procedures, retrieves data values from cross-referenced data sources joined dynamically with the JOIN command. The function is valid in both MODIFY COMPUTE and VALIDATE commands.

The syntax for the LOOKUP function is

```
rfield/I1 = LOOKUP(field);
```

where:

rfield

Contains the return code (1 or 0) after the LOOKUP function executes.

field

Is the name of any field in a cross-referenced data source. After the LOOKUP, this fieldname contains the field's value for you to use as needed.

To use this feature most efficiently with Adabas, specify a cross-referenced field for which an Adabas descriptor has been established.

Note:

- The LOOKUP function is not supported between Adabas data sources and FOCUS databases in either direction.

- The extended syntax of the LOOKUP function (parameters GE and LE) is not valid for Adabas data sources. LOOKUP can only retrieve values that match exactly. Refer to the FOCUS for S/390 documentation for more information.

The FIND Function

The FIND function, used with COMBINE structures in FOCUS MODIFY procedures or with any file in a Maintain procedure, verifies the existence of a value in another data source. The FIND function sets a temporary field to 1 if the value exists in the other data source and to 0 if it does not. FIND does not return any actual data values.

Use FIND only with a data source referenced in a COMBINE command or a MAINTAIN FILE command. With COMBINE, if the FIND is for a field in a VSAM file, this field must be the index or alternate index field. For Maintain, the field must be indexed only if it is in a FOCUS database.

The syntax for the FIND function is

```
rfield/I1 = FIND(fieldname AS dbfield IN file);
```

where:

rfield

Contains the return code (1 or 0) after the FIND function executes.

fieldname

Is the comparison field from one COMBINE data source or one data source referenced in a MAINTAIN FILE command.

dbfield

For MODIFY, is the field in another COMBINE file structure or an indexed field in a VSAM file, to use for the value comparison. The AS dbfield clause is optional if rfield and dbfield have the same name.

For Maintain, is a fieldname from one of the files listed in the MAINTAIN FILE command, qualified with its file name.

To use this feature most efficiently with Adabas, specify a field for which an Adabas descriptor has been established.

file

In MODIFY, names the data source or VSAM file in which dbfield resides. In Maintain, is ignored.

The FIND function is only supported within MODIFY or Maintain procedures. For more information, consult the FOCUS for S/390 documentation.

Data Adapter Error Messages and Adabas Response Codes

This section lists the Adabas Write Data Adapter error messages and common Adabas response codes.

Adabas Write Data Adapter Error Messages

In the following error messages, the term *number* refers to the response code that Adabas returns. For example, the following message references Adabas error response code 113 in the Adabas Messages and Codes manual:

```
(FOC4561)       ERROR IN ADABAS INCLUDE/113
```

The following messages apply to the Adabas Write Data Adapter:

```
(FOC4555)       INVALID UNQKEYNAME FOR SEGMENT
```

(FOC4557) **ERROR IN ADABAS COMMIT/***number*

(FOC4558) **ERROR IN ADABAS ROLLBACK/***number*

(FOC4559) **ERROR IN ADABAS UPDATE/***number*

(FOC4560) **ERROR IN ADABAS DELETE/***number*

(FOC4561) **ERROR IN ADABAS INCLUDE/***number*

(FOC4562) **RECORD IS HELD BY ANOTHER USER**

(FOC4563) **INCORRECT INCLUDE REQUEST FOR NON UNIQUE KEY**

(FOC4564) **THIS OCCURRENCE ALREADY EXISTS. USE UPDATE COMMAND.**

(FOC4565) **IGNORED ATTEMPT TO CHANGE NONUPDATABLE FIELD**

An UPDATE command was used against a nonupdatable field, i.e. - field described in AFD as SUB/SUPERDESCRIPTOR or part of it;
- field that refers in MFD to the same DB field (ALIAS) as another field (synonym case).

For individual Adabas Data Adapter error messages, consult Appendix B, *Adabas Interface Error Messages* in the FOCUS for IBM Mainframe Adabas Interface User's Manual and Installation Guide.

Adabas Response Codes

If variables FOCERROR or RETCODE have the value 999, the Adabas

FOCUS Error	Adabas Response Code	Description
FOC4490	148	The database is down or unavailable.
FOC4496	148	Check the DBNO in the Access File.
FOC4500	113	The FILENO for the segment in the Access File is not correct.
FOC4504	0	Check the DBNO and FILENO in both the PREDDB and PREDEL Access Files.
FOC4504	17	The database contains no data and/or the database does not exist in the Predict dictionary.
FOC4504	28	Check the compatibility of the Predict dictionary and FDT listing.
FOC4504	41	The Predict dictionary and FDT may not match. Check the group in the FDT. It may have been created differently in the Predict dictionary.
FOC4504	201	The password is not correct.

FOCUS Error	Adabas Response Code	Description
Any	9	As a rule: The transaction time limit (TT) or the transaction non-activity time limit (TNAE, TNAX, TNAA) has been exceeded. For details see a subcode value in the Adabas control block's ADD2 field in the data adapter's level 3 trace, and refer to its explanation in the Adabas *Messages and Codes* manual.
Any	47	The maximum value for the NISNHQ parameter was exceeded. Too many records are held. You can release them by periodically issuing the COMMIT command.
Any	40-45	One or more errors occurred in the Format Buffer. There is a discrepancy between the Master File and the Adabas FDT.

FOCUS Error	Adabas Response Code	Description
Any	50-59	One or more errors occurred in the Record Buffer. There is a discrepancy between the Master File and the Adabas FDT.
Any	145	An attempt was made to hold a record already in the hold queue for another user.

response code was not used. Use the data adapter error message to determine the cause of the error.

For details on the Adabas response codes that may accompany the Adabas Data Adapter error messages, see your Software AG publication, *General Reference Set: Messages and Codes Manual*. If a subcode details the specific cause and actions for the response code in this manual, you can find the subcode's value in the low-order (rightmost) two bytes of the Adabas control block's ADD2 field using the Adabas Data Adapter's level 3 trace. The trace facility for the Write data adapter is the same as for the Read data adapter.

Common User Errors

1. Using a group field in an INCLUDE or UPDATE command.

   In the following request, the record is not inserted because the field list contains a group field (LEAVE_DATA):

```
MODIFY FILE EMPWRITE
-* This is an example of the error caused by using a
-* group field (LEAVE_DATA), rather than only elementary fields,
-* in an INCLUDE process.
-* The insert (Segname=S01) is rejected with the error -
-* (FOC4491) FIND ERROR READING FIRST RECORD  : S01     /61
-*
-* Insert for SEGNAM=S01     - ACCESS=ADBS
   COMPUTE EMPLOYEE_ID = 'EMPID003';
   MATCH EMPLOYEE_ID
   ON MATCH COMPUTE
      LAST_NAME   =  'SMITH'     ;
      FIRST_NAME  =  'ROBERT'    ;
      MIDDLE_NAME =  'EDWARD'    ;
      CITY        =  'NEW YORK' ;
      ZIP_CODE    =  '10121'     ;
      COUNTRY     =  'USA'       ;
      LEAVE_DATA  =  'VAC0801080700';
   ON MATCH INCLUDE
   ON NOMATCH REJECT
DATA
END
```

   Solution: Use only elementary fields in the field list for INCLUDE and UPDATE.

2. Issuing an INCLUDE command for a new key value with additional fields in the MATCH list.

In the following example EMPLOYEE_ID is defined as a unique key in the Access File. However, Adabas may insert the record even if the employee ID value already exists in the data source:

```
MODIFY FILE EMPWRITE

-* INSERT FOR SEGNAM=S01 - ACCESS=ADBS
   COMPUTE EMPLOYEE_ID = 'EMPID001';
           LAST_NAME =   'SMITH';
   MATCH EMPLOYEE_ID LAST_NAME
   ON NOMATCH COMPUTE
     FIRST_NAME = 'ROBERT';
     MNAME      = 'EDWARD';
     CITY       = 'NEW YORK';
  ON NOMATCH INCLUDE
  ON MATCH REJECT
DATA
END
```

Solution: Issue the MATCH command for the field EMPLOYEE_ID alone, then enter values into the other fields and issue the INCLUDE command.

7.0.9 New Features

Fusion

NF575: Fusion

General Enhancements

NF716: Euro Currency Support
NF655: FOCPROF - The System Wide Profile
NF735: Enhancement to ? SET
NF746: Leading Zeros
NF656: Controlling REBUILD Messages
NF740: Changes to the REBUILD Prompt
NF660: Multi-volume Support in MVS FOCUS
NF670: DYNAM Support for Unit Count
NF718: DYNAM Support for Existing Relative GDG Numbers
NF745: ? PTF Enhancements

Reporting Enhancements

NF691: Escape Character for LIKE Predicate
NF730: Hold Format PDF
NF744: HOLD FORMAT EXCEL
NF748: HOLD FORMAT WP With Carriage Control

Performance Enhancements

NF654: HOLD From External Sort
NF597: Aggregation by External Sort
NF728: Changing Retrieval Order with Aggregation

Data Adapter Enhancements

NF584: Dynamically Setting the IDMS DBNAME and DICTNAME
NF652: Teradata Interface Kanji Support
NF673: Model 204 Interface Account Split
NF720: SQLJOIN OUTER Setting for Relational Interfaces

FOCUS Client

NF722: FOCUS Client DNS Names Support

NF575: Fusion

FOCUS Version 7.0 release 9 introduces the Fusion database for CMS and MVS FOCUS. Fusion is a high performance database whose unique Multi Dimensional Indexing (MDI) architecture extends the scope of high-speed multi-dimensional query performance.

For complete details, see the *Fusion User's Manual for EDA 4.2 and FOCUS 7.0 (*DN3700041.1198).

NF716: Euro Currency Support

With the introduction of the euro currency, businesses need to maintain books in two currencies, add new fields to their database designs, and perform new types of currency conversions. This new feature gives FOCUS the ability to perform currency conversions according to the rules specified by the European Union. Before you can use FOCUS to process currency conversions, you must:

- Create a currency database with the currency IDs and exchange rates you will use. See *Creating the Currency Database*.
- Identify fields in your data sources that represent currency data. See *Identifying Fields That Contain Currency Data.*
- Activate your currency database. See *Activating the Currency Database*.

After you complete these preliminary steps, you can perform currency conversions. See *Processing Currency Data*.

Note: Operating system vendors are in the process of integrating the euro currency symbol into their environments. As the euro symbol becomes available, FOCUS will support it.

Converting Currencies

Although the euro was introduced in 11 countries of the European Union on January 1, 1999, it will not immediately replace local currencies in those countries. During the transition period from 1999 to 2002, both traditional currencies and the euro will be used simultaneously for accounting purposes and non-cash transactions in each participating country. Euro cash will not be introduced until January 1, 2002, and by July 1, 2002 the traditional currencies will no longer be legal tender.

On the last day of 1998, the European Union set fixed exchange rates between the euro and the traditional national currency in each of the 11 adopting member nations. While the exchange rates within “Euroland” will remain fixed, exchange rates between the euro and non-euro countries will continue to vary freely and, in fact, several rates may be in use at one time (for example, actual and budgeted rates).

The European Union has established the following rules for currency conversion:

- The exchange rate must be specified as a decimal value, r, with six significant digits (not six decimal places). For example, 123.456 has six significant digits but not six decimal places. This rate will establish the following relationship between the euro and the particular national currency:

  1 euro = r national units
- To convert from the euro to the national unit, multiply by r and round the result to two decimal places.
- To convert from the national currency to the euro, divide by r and round the result to two decimal places.

- To convert from one national currency to another, first convert from one national unit to the euro, rounding the result to at least three decimal places (FOCUS rounds to exactly three decimal places). Then convert from the euro to the second national unit, rounding the result to two decimal places. The following diagram illustrates this two-step conversion process known as *triangulation*:

Preparing FOCUS to Process Currency Conversions

Although 11 or more currencies in the European Union will be converting to the euro, more than 100 currencies have a recognized status worldwide. In addition, you may need to define custom currencies for some applications.

You identify your currency codes and rates by creating a currency database. The currency database can be any type of data source that FOCUS can access.

Creating the Currency Database

For each type of currency you need, you must supply the following values in your currency database:

- A three-character code to identify the currency, such as USD for U.S. dollars or BEF for Belgian francs. (For a partial list of recognized currency codes, see *Sample Currency Codes*.)
- One or more exchange rates for the currency.

There is no limit to the number of currencies you can add to your currency database; the currencies you can define are not limited to official currencies and, therefore, the currency database can be fully customized for your applications.

We strongly recommend that you create a separate database for the currency data rather than adding the currency fields to another data source. A separate currency database enhances performance and minimizes resource utilization because FOCUS loads the currency database into memory before you perform currency conversions.

Syntax How to Specify Currency Codes and Rates in a Master File

The currency database can be any type of data source accessible by FOCUS (for example, FOCUS, FIX, DB2, or VSAM). The currency Master File must have one field that identifies each currency ID you will use and one or more fields to specify the exchange rates.

The syntax is

```
FIELD  = CURRENCY_ID,   FORMAT =  A3,                      ACTUAL = A3 ,$
FIELD  = rate1,         FORMAT = {D12.6|numeric_format1}, ACTUAL = A12,$
    .
    .
    .
FIELD  = raten,         FORMAT = {D12.6|numeric_formatn}, ACTUAL = A12,$
```

where:

`CURRENCY_ID`

Is the required field name. The values stored in this field are the three-character codes that identify each currency, such as USD for U.S. dollars. Each currency ID can be a universally recognized code or a user-defined code. **Note:** FOCUS automatically recognizes the code EUR; you should *not* store this code in your currency database. See *Sample Currency Codes* for a list of common currency codes.

`rate1,...,raten`

Are types of rates (such as BUDGET, FASB, ACTUAL) to be used in currency conversions. Each rate is the number of national units that represent one euro.

`numeric_format1,...,numeric_formatn`

Are the display formats for the exchange rates. Each format must be numeric. The recommended format, D12.6, ensures that the rate is expressed with six significant digits as required by the European Union conversion rules. Do not use Integer format (I).

`ACTUAL An`

Is required only for non-FOCUS data sources.

Note: The maximum number of fields in the currency database must not exceed 255 (that is, the CURRENCY_ID field plus 254 currency conversion fields).

Example Specifying Currency Codes and Rates in a Master File

The following Master File for a comma-delimited currency database specifies two rates for each currency, ACTUAL and BUDGET:

```
FILE = CURRCODE, SUFFIX = COM,$
FIELD = CURRENCY_ID,      FORMAT = A3,     ACTUAL = A3 ,$
FIELD = ACTUAL, ALIAS =, FORMAT = D12.6, ACTUAL = A12 ,$
FIELD = BUDGET, ALIAS =, FORMAT = D12.6, ACTUAL = A12 ,$
```

The following is sample data for the currency database defined by this Master File:

```
FRF,  6.55957,  6.50000,$
USD,  1.17249,  1.20000,$
BEF, 40.3399, 41.00000,$
```

Identifying Fields That Contain Currency Data

Once you have created your currency database, you must identify the fields in your data sources that represent currency values. To designate a field as a currency-denominated value (a value that represents a number of units in a specific type of currency) add the CURRENCY attribute to one of the following:

- The FIELD specification in the Master File.
- The left side of a DEFINE or COMPUTE.

Syntax How to Identify a Currency Value

Use the following syntax to identify a currency-denominated value:

- In a Master File

```
FIELD = currfield, FORMAT = numeric_format, ,
CURR = {curr_id|codefield} ,$
```

- In a DEFINE in the Master File

```
DEFINE currfield/numeric_format CURR curr_id = expression ;$
```

- In a DEFINE FILE command

```
DEFINE FILE filename
currfield/numeric_format CURR curr_id = expression ;
END
```

- In a COMPUTE command

```
COMPUTE currfield/numeric_format CURR curr_id = expression ;
```

where:

filename

Is the name of the file for which this field is defined.

currfield

Is the name of the currency-denominated field.

numeric_format

Is a numeric format. Depending on the currency denomination involved, the recommended number of decimal places is either two or zero. Do not use I or F format.

CURR

Indicates that the field value represents a currency-denominated value. CURR is an abbreviation of CURRENCY, which is the full attribute name.

curr_id

Is the three-character currency ID associated with the field. In order to perform currency conversions, this ID must either be the value EUR or match a CURRENCY_ID value in your currency database.

codefield

Is the name of a field, qualified if necessary, that contains the currency ID associated with *currfield*. The code field should have format A3 or longer and is interpreted as containing the currency ID value in its first three bytes.

For example:

```
FIELD = PRICE, FORMAT = P12.2C, ..., CURR = TABLE.FLD1,$
.
.
.
FIELD = FLD1, FORMAT = A3, ...,$
```

The field named FLD1 contains the currency ID for the field named PRICE.

expression

Is a valid expression.

Example Identifying a Currency-denominated Field

Assume that the currency database contains the currency ID value BEF (Belgian francs).

If the FINANCE data source contains a field named PRICE that is denominated in Belgian francs, the description of PRICE in the FINANCE Master File could be:

```
FIELD = PRICE, ALIAS=, FORMAT = P17.2, CURR=BEF,$
```

Activating the Currency Database

Before you can perform currency conversions, you must bring the relevant currency database into memory by issuing the SET EUROFILE command.

Syntax How to Activate Your Currency Database

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

```
SET EUROFILE = {ddname|OFF}
```

where:

ddname

Is the name of the Master File for the currency database. There is no default value for EUROFILE. The ddname must refer to a data source known to FOCUS and accessible by FOCUS in read-only mode.

OFF

Deactivates the currency database and removes it from memory.

During your FOCUS session, if you want to access a different currency database, you can re-issue the SET EUROFILE command.

Note:

- You cannot append any additional SET parameters to the SET EUROFILE command line. For example, the PAUSE setting would be lost if you issued the following command:

  ```
  SET EUROFILE=filename , PAUSE=OFF
  ```

- You cannot issue the SET EUROFILE command within a TABLE request.

Syntax How to Determine the Currency Database in Effect

If you want to determine which currency database is in effect, issue the ? SET ALL command or the new EUROFILE query command:

```
? SET EUROFILE
```

Example Determining the Currency Database in Effect

Assume the currency database is named CURRCODE.

If you issue the following commands:

```
set eurofile = currcode
? set eurofile
```

FOCUS returns the following response:

```
EUROFILE            CURRCODE
```

Reference SET EUROFILE Error Messages and Notes

Issuing the SET EUROFILE command when the currency database Master File does not exist generates the following error message:

(FOC205) THE DESCRIPTION CANNOT BE FOUND FOR FILE NAMED: *ddname*

Issuing the SET EUROFILE command when the currency Master File specifies a FOCUS database and the associated FOCUS database does not exist generates the following error message:

(FOC036) NO DATA FOUND FOR THE FOCUS FILE NAMED: *name*

Note for Pooled Table users: The SET EUROFILE command creates a pool boundary.

Processing Currency Data

After you have created your currency database, identified the currency-denominated fields in your data sources, and activated your currency database, you can perform currency conversions.

Each currency ID in your currency database generates a virtual conversion function whose name is the same as its currency ID. For example, if you added BEF to your currency database, a virtual BEF currency conversion function will be generated.

The euro function, EUR, is supplied automatically with FOCUS. You do not need to add the EUR currency ID to your currency database.

Syntax How to Convert Currency Data

Use the following syntax for calling a currency conversion function

- In a TABLE, GRAPH, or MODIFY procedure:

```
DEFINE FILE filename
result/format [CURR curr_id] = curr_id(infield, rate1 [,rate2]);
END
```

or

```
COMPUTE result/format [CURR curr_id] = curr_id(infield, rate1
[,rate2]);
```

- In a Master File:

```
DEFINE result/format [CURR curr_id] = curr_id(infield, rate1
[,rate2]);$
```

where:

filename

Is the name of the file for which this field is defined.

result

Is the converted currency value.

format

Must be a numeric format. Depending on the currency denomination involved, the recommended number of decimal places is either two or zero. Do not use I or F format. The result will always be rounded to two decimal places, which will display if the format allows at least two decimal places.

curr_id

Is the currency ID of the result field. This ID must be the value EUR or match a currency ID in your currency database; any other value generates the following message

```
(FOC263)  EXTERNAL FUNCTION OR LOAD MODULE NOT FOUND: curr_id
```

Note: The CURR attribute on the left side of the DEFINE or COMPUTE identifies the result field as a currency-denominated value which can be passed as an argument to a currency function in subsequent currency calculations. Adding this attribute to the left side of the DEFINE or COMPUTE does not invoke any format or value conversion on the calculated result.

infield

Is a currency-denominated value. This input value will be converted from its original currency to the *curr_id* denomination. If the *infield* and *result* currencies are the same, no calculation is performed and the *result* value is the same as the *infield* value.

`rate1`

Is the name of a rate field from the currency database. The *infield* value is divided by its currency's *rate1* value to produce the equivalent number of euros.

If *rate2* is not specified in the currency calculation and triangulation is required, this intermediate result is then multiplied by the *result* currency's *rate1* value to complete the conversion.

In certain cases, you may need to provide different rates for special purposes. In these situations you can specify any field or numeric constant for *rate1* as long as it indicates the number of units of the *infield* currency denomination that equals one euro.

`rate2`

Is the name of a rate field from the currency database. This argument is only used for those cases of triangulation in which you need to specify different rate fields for the *infield* and *result* currencies. It is ignored if the euro is one of the currencies involved in the calculation.

The number of euros that was derived using *rate1* is multiplied by the *result* currency's *rate2* value to complete the conversion.

In certain cases, you may need to provide different rates for special purposes. In these situations you can specify any field or numeric constant for *rate2* as long as it indicates the number of units of the *result* currency denomination that equals one euro.

Note: MAINTAIN does not support these currency conversion functions.

Example Converting Currencies

Assume that the currency database contains the currency IDs USD and BEF, and that PRICE is denominated in Belgian francs as follows:

```
FIELD = PRICE, ALIAS=, FORMAT = P17.2, CURR=BEF,$
```

- The following example converts PRICE to euros and stores the result in PRICE2 using the BUDGET conversion rate for the BEF currency ID:

  ```
  COMPUTE PRICE2/P17.2 CURR EUR = EUR(PRICE, BUDGET);
  ```

- This example converts PRICE from Belgian francs to US dollars using the triangulation rule:

  ```
  DEFINE PRICE3/P17.2 CURR USD = USD(PRICE, ACTUAL);$
  ```

  First PRICE is divided by the ACTUAL rate for Belgian francs to derive the number of euros rounded to three decimal places. Then this intermediate value is multiplied by the ACTUAL rate for US dollars and rounded to two decimal places.

- The following example uses a numeric constant for the conversion rate:

  ```
  DEFINE PRICE4/P17.2 CURR EUR = EUR(PRICE,5);$
  ```

- The next example uses the ACTUAL rate for Belgian francs in the division and the BUDGET rate for US dollars in the multiplication:

  ```
  DEFINE PRICE5/P17.2 CURR USD = USD(PRICE, ACTUAL, BUDGET);$
  ```

Reference Currency Calculation Processing and Messages

The result is always calculated with very high precision, 31 to 36 significant digits, depending on platform. The precision of the final result is always rounded to two decimal places. In order to display the result to the proper precision, its format must allow at least two decimal places.

Issuing a TABLE request against a Master File that specifies a currency code not listed in the active currency database generates the following message:

```
(FOC1911) CURRENCY IN FILE DESCRIPTION NOT FOUND IN DATA
```

A syntax error or undefined fieldname in a currency conversion expression generates the following message:

```
(FOC1912)  ERROR IN PARSING CURRENCY STATEMENT
```

Reference Sample Currency Codes

The following rates were in effect on December 31, 1998. Euroland countries as of that date are marked with an asterisk (*). Their rates are fixed and will not change; the rates for other countries can change over time:

Country	Currency Code	Rate
Austria*	ATS	13.7603
Belgium*	BEF	40.3399
Canada	CAD	1.7978
Denmark	DKK	7.46215
European Union	EUR	1
Finland*	FIM	5.94573
France*	FRF	6.55957
Germany*	DEM	1.95583

Country	Currency Code	Rate
Greece	GRD	328.6
Ireland*	IEP	0.787564
Italy*	ITL	1936.27
Japan	JPY	133.149
Luxembourg*	LUF	40.3399
Netherlands*	NLG	2.20371
Norway	NOK	8.91039
Portugal*	PTE	200.482
Spain*	ESP	166.386
Sweden	SEK	9.52669
Switzerland	CHF	1.61093
UK	GBP	0.706739
USA	USD	1.17249

Example Converting U.S. Dollars to Euros, French Francs, and Belgian Francs

Assume PRICE is denominated in U.S. dollars and ACTUAL is the name of a rate in the currency database. Using the currency conversion rates from *Sample Currency Codes*, the following FOCEXEC converts PRICE to euros, French francs, and Belgian francs:

```
-* CURRCODE IS THE CURRENCY DATABASE
-* CURRDATA IS THE DATA SOURCE WITH CURRENCY-DENOMINATED FIELDS

-* THE FOLLOWING FILEDEFS ARE FOR RUNNING UNDER CMS
CMS FILEDEF CURRCODE DISK CURRCODE TEXT A
CMS FILEDEF CURRDATA DISK CURRDATA TEXT A

-* THE FOLLOWING ALLOCATIONS ARE FOR RUNNING UNDER MVS
-* DYNAM ALLOC FILE CURRCODE DA USER1.FOCEXEC.DATA(CURRCODE) SHR REU
-* DYNAM ALLOC FILE CURRDATA DA USER1.FOCEXEC.DATA(CURRDATA) SHR REU

SET EUROFILE = CURRCODE

DEFINE FILE CURRDATA
PRICEEUR/P17.2 CURR EUR = EUR(PRICE, ACTUAL);
END

TABLE FILE CURRDATA
PRINT PRICE PRICEEUR AND COMPUTE
PRICEFRF/P17.2 CURR FRF = FRF(PRICE, ACTUAL);
PRICEBEF/P17.2 CURR BEF = BEF(PRICE, ACTUAL);
END
```

This request generates the following report:

```
PAGE     1


            PRICE          PRICEEUR          PRICEFRF          PRICEBEF
            -----          --------          --------          --------
             5.00              4.26             27.97            172.01
             6.00              5.12             33.57            206.42
            40.00             34.12            223.78           1376.20
            10.00              8.53             55.95            344.06
```

Note: You cannot use the derived euro value PRICEEUR in a conversion from USD to BEF. PRICEEUR has two decimal places (P17.2), not three, as the triangulation rules require. Therefore, PRICEEUR yields the following inaccurate result (see PRICEBEF above) and is not valid as the intermediate value in a currency conversion that requires triangulation:

```
COMPUTE PRICENEW/P17.2 CURR BEF = BEF(PRICEEUR, ACTUAL);
 PRICENEW
 --------
   171.85
   206.54
  1376.40
   344.10
```

NF744: HOLD FORMAT EXCEL

FOCUS can now format report output as a Microsoft® Excel spreadsheet. When you use the HOLD FORMAT EXCEL syntax, FOCUS creates a binary file containing all columns of the report output with their column headings.

If you are using the Web Interface for FOCUS, the Excel spreadsheet can be downloaded automatically to your Web Browser. If you do not have the Web Interface, you can transfer the spreadsheet file to any environment that supports Excel, such as a PC running Microsoft Windows®; the recommended transfer protocol is FTP in binary mode.

You can also store the Excel file on a web server where users can display it using any browser configured with Excel as a plug-in.

Syntax How to Create an Excel Spreadsheet

The syntax is.

```
[ON TABLE] HOLD [AS filename] FORMAT EXCEL
```

where:

`ON TABLE`

Is required syntax if you create the Excel file within a report request.

`filename`

Assigns a name to the Hold file. The default name is HOLD.

In MVS, unless you allocate this ddname to a permanent file, FOCUS allocates it to a temporary data set.

In CMS, this name becomes the file name. The file type is XLS.

By default, the extension for Microsoft Excel files is .xls in environments that support file names with extensions. Therefore, the file name after transfer is filename.xls.

Syntax How to Create and Download an Excel File With the Web Interface for FOCUS

If you are using the Web Interface for FOCUS, you can create the Excel spreadsheet and automatically download it to your Web Browser with the following syntax:

```
[ON TABLE] PCHOLD FORMAT EXCEL
```

where:

```
ON TABLE
```

Is required syntax if you create the Excel file within a report request.

On the PC, the name of the Excel spreadsheet file is *hold.xls*.

For a detailed description of how to download report output using the Web Interface for FOCUS and PCHOLD, see *NF730: Hold Format PDF.*

Example Creating an Excel Spreadsheet in a Report Request

Consider the following request run in CMS:

```
TABLE FILE CAR
PRINT CAR MODEL RETAIL_COST DEALER_COST
BY COUNTRY
ON TABLE HOLD AS EXCEL1 FORMAT EXCEL
END
```

This request creates the file EXCEL1 XLS A. After transferring this file (using FTP in binary mode) to a file named *excel1.xls* on your PC, you can open it in Excel:

	A	B	C
1	COUNTRY	CAR	MODEL
2	ENGLAND	JAGUAR	V12XKE AUTO
3	ENGLAND	JAGUAR	XJ12L AUTO
4	ENGLAND	JENSEN	INTERCEPTOR III
5	ENGLAND	TRIUMPH	TR7
6	FRANCE	PEUGEOT	504 4 DOOR
7	ITALY	ALFA ROMEO	2000 GT VELOCE
8	ITALY	ALFA ROMEO	2000 SPIDER VELOCE
9	ITALY	ALFA ROMEO	2000 4 DOOR BERLINA
10	ITALY	MASERATI	DORA 2 DOOR
11	JAPAN	DATSUN	B210 2 DOOR AUTO
12	JAPAN	TOYOTA	COROLLA 4 DOOR DIX AUT
13	W GERMANY	AUDI	100 LS 2 DOOR AUTO
14	W GERMANY	BMW	2002 2 DOOR
15	W GERMANY	BMW	2002 2 DOOR AUTO

NF730: Hold Format PDF

FOCUS can now generate output in Adobe® Portable Document Format (PDF). This feature enables Web Interface users to produce reports with all PDF formatting options (for example, headings, footings, and titles) correctly aligned on the physical pages. PDF also supports StyleSheets incorporating drill-downs and links to URLs.

Required Software Configuration

Adobe Acrobat Reader® Version 3.01 or higher is required to display PDF output, however, no third-party products are needed to *produce* it. The Adobe Acrobat Reader is Internet shareware for Windows 95, Windows NT, UNIX, and Macintosh, and is available free from their Web site, http://www.adobe.com.

Reports viewed with Adobe Acrobat Reader look precisely as if printed. By configuring the Acrobat Reader as a plug-in on your Web browser, PDF output displays directly inside the browser window in printed format without additional setup or configuration.

Browser users who have not installed Acrobat Reader can save PDF files to disk or download them to a PC when prompted by the browser and then transfer them later to a machine with the Acrobat Reader to display them. Users can then either run the standalone Acrobat Reader program or use a browser with the PDF plug-in to view the PDF files.

Syntax How to Create PDF Output

The syntax is:

```
ON TABLE {HOLD|SAVE} FORMAT PDF [AS filename]
```

where:

`ON TABLE`

Is required syntax if you create the PDF file within a report request.

`filename`

Assigns a name to the Hold file. The default name is HOLD. If you specify an optional AS filename the name that you supply (1-8 characters) is used in place of HOLD. If you issue a FILEDEF for the filename, the PDF output is created in the file specified in the FILEDEF.

In MVS, unless you allocate this ddname to a permanent file, FOCUS allocates it to a temporary dataset.

In CMS, this name becomes the filename. The filetype is PDF.

Downloading PDF Output

The technique for downloading PDF files depends on how you accessed FOCUS. PDF-formatted Hold files created on the mainframe can be downloaded to the PC using FTP transfers. With the Web Interface, browser users can issue the following PCHOLD syntax to download PDF files to their browsers.

```
ON TABLE PCHOLD FORMAT PDF
```

The Web Interface automatically FTPs output to your PC if you choose. A browser window opens after the command is issued, inquiring whether you wish to have the file opened immediately or saved to disk.

You have chosen to download a file from this location.
webtoe916f4b18. from ibivm
What would you like to do with this file?
Open this file from its current location
Save this file to disk
Always ask before opening this type of file
OK
Cancel
More Info

Example Creating a PDF-formatted Report Using the CAR File

This report request generates the PDF report that follows.

```
TABLE FILE CAR
HEADING CENTER
"CAR COSTS PER COUNTRY"
FOOTING
"---> Uncentered footing"
PRINT
MODEL DEALER_COST RETAIL_COST
BY COUNTRY
BY CAR
WHERE COUNTRY = 'ENGLAND' OR 'ITALY'
ON CAR SUBTOTAL
ON TABLE PCHOLD FORMAT PDF
END
```

PAGE 1

CAR COSTS PER COUNTRY

COUNTRY	CAR	MODEL	DEALER_COST	RETAIL_COST
ENGLAND	JAGUAR	V12XKE AUTO	7,427	8,878
		XJ12L AUTO	11,194	13,491
*TOTAL CAR JAGUAR			18,621	22,369
	JENSEN	INTERCEPTOR III	14,940	17,850
*TOTAL CAR JENSEN			14,940	17,850
	TRIUMPH	TR7	4,292	5,100
*TOTAL CAR TRIUMPH			4,292	5,100
ITALY	ALFA ROMEO	2000 GT VELOCE	5,660	6,820
		2000 SPIDER VELOCE	5,660	6,820
		2000 4 DOOR BERLINA	4,915	5,925
*TOTAL CAR ALFA ROMEO			16,235	19,565
	MASERATI	DORA 2 DOOR	25,000	31,500
*TOTAL CAR MASERATI			25,000	31,500
TOTAL			79,088	96,384

---> UNCENTERED FOOTING

NF654: HOLD From External Sort

External sorts can be used to create HOLD files. This can lead to savings of up to twenty percent on processing time. The gains are most notable with relatively simple requests against large databases.

Syntax How to Create HOLD FILES with an External Sort

```
SET EXTHOLD = [OFF|ON]
```

where:

`OFF`

Disables HOLD files by an external sort. OFF is the default.

`ON`

Enables HOLD files by an external sort.

Conditions for Using External Sort to Create a HOLD File

- The default setting of EXTSORT=ON must be in effect.
- EXTHOLD must be ON.
- Request must contain a BY field.
- Request must contain ON TABLE HOLD or ON TABLE HOLD AS.

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

NF597: Aggregation by External Sort

External sorts can be used to perform aggregation with a significant decrease in processing time in comparison to using the sort facility of FOCUS. The gains will be most notable with relatively simple requests against large databases.

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 data fields present.

`ON`

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

Conditions for Aggregating with an External Sort

- You must be using SYNCSORT or DFSORT.
- EXTAGGR cannot be set to OFF.
- Your query should be simple. (AUTOTABLEFable)
- 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 COMPUTEd 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 by the FOCUS sort.

Example How Using an External Sort for Aggregation Can Change Your Output

Using an external sort for aggregation can change the output of your report request.

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

With aggregation in the external sort turned on:

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

The output is:

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

With aggregation in the external sort turned off:

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

The output is:

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

Note: The SET SUMPREFIX command in conjunction with aggregation using an external sort also affects the order of information displayed in your report. For complete information on SUMPREFIX please see New Feature Bulletin 728, SUMPREFIX.

Reference Special Considerations

When aggregation is performed by an external sort the statistical variables &RECORDS and &LINES are of equal value. This is done 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.

NF728: Changing Retrieval Order with Aggregation

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

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

The SUMPREFIX command deals with this difference in behavior by allowing users to choose which order the answer set should display.

Syntax Setting Retrieval Order

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

where:

`LST`

Displays the last value in cases of data aggregation of alphanumeric or smart date data types.

`FST`

Displays the first value in cases of data aggregation of alphanumeric or smart date data types.

NF655: FOCPROF - The System Wide Profile

FOCPROF is a new global profile for FOCUS. While the FOCPARM profile supports only FOCUS SET commands, this new profile can contain any command that is valid in a FOCEXEC, including TABLE, GRAPH, MATCH, MODIFY, MAINTAIN, REBUILD, COMPILE, LOAD, Dialogue Manager commands, CMS commands, TSO commands, and DYNAM commands.

FOCUS Profiles

With the addition of FOCPROF, there are now three FOCUS profiles: FOCPARM, FOCPROF, and PROFILE.

The files FOCPARM and FOCPROF:

- Are members of the ERRORS PDS on MVS.
- Have filetype ERRORS on CMS.

The file PROFILE:

- Is a member of the FOCEXEC PDS on MVS.
- Has filetype FOCEXEC on CMS.

Using FOCUS Profiles

The order of execution of FOCUS profiles is:

1. FOCPARM, which can contain FOCUS SET commands only.
2. FOCPROF, the new global profile.
3. PROFILE.

ProcedureHow to Create a FOCPROF Profile

- For MVS, create a new member of the ERRORS PDS named FOCPROF.
- For CMS, create a file with filename FOCPROF and filetype ERRORS.

Edit the FOCPROF file to contain the commands to be executed each time FOCUS is invoked.

NF660: Multi-volume Support in MVS FOCUS

The latest release of MVS FOCUS gives sites the option of allocating FOCUS databases, Fusion databases, and FOCUS-created sequential files across multiple volumes.

Advantages of Multi-volume Data Sources

Many sites prefer to distribute high volume data sources across multiple volumes in order to manage:

- Use of storage on specific devices or device types.
- Run-time access to these devices.
- B37 abends.

You can now use this performance tuning technique (also known as *data striping*) with FOCUS databases, Fusion databases, and FOCUS-created sequential files in the MVS batch, TSO, and MSO environments.

Allocating Multi-volume Data Sources

The SPACE parameter for allocating a data source can include a primary and a secondary allocation. The primary allocation is the amount of space allocated the first time data is written to the data set. The secondary allocation is the amount of space to be allocated, when necessary, for up to 15 additional extents.

For a single-volume data source, processing terminates with a B37 abend when the system detects either of the following conditions:

- A need for more than 16 extents.

- A need for a new secondary extent (below the 16-extent limit) when enough space is not available on the volume.

FOCUS returns the following message to indicate that one of these conditions has occurred:

```
(FOC198) FATAL ERROR IN DATABASE I/O. FOCUS TERMINATING CODE: 00000070
```

You can prevent this type of abnormal termination by allocating multiple volumes to the data source. With multiple volumes, an out of space condition on the first volume causes allocation to start on another volume.

With multiple volumes, the allocation process varies slightly for each of the following:

- The first volume.
- Intermediate volumes.
- The last volume.

The following table describes the multi-volume allocation process:

Primary Allocation	The primary allocation is applied to the first volume only. It can consist of the number of extents allowed by MVS for a primary allocation. **Note:** A data source with no secondary space allocation is limited to a single volume.

Secondary Allocation	**1.First volume:** As many extents as are available, up to the 16-extent limit, are allocated and filled before continuing to the second volume. **2.Intermediate volume**: Depending on the space available, up to 16 extents are filled before allocation begins on the next volume. **3.Last volume:** Once the need for a number of extents greater than the limit is detected: • For a FOCUS or Fusion database, processing terminates with the following message `(FOC198) FATAL ERROR IN DATABASE I/O. FOCUS TERMINATING CODE 00000070` • For temporary FOCSORT files, after all volumes and extents are filled allocation spills to up to 15 additional temporary files, each with 16 extents. (This feature, NF536, *Multi-Image FOCSORT*, was added in FOCUS 7.0.6.) The SPACE allocation for each spill file is the same as the SPACE allocation for the original FOCSORT file.

	For example: `//FOCSORT DD SPACE=(TRK,(5,5))` This allocates a total of 5 + (5*15) = 80 tracks. When the 81st track is needed, another temporary data set is allocated with the parameter SPACE=(TRK,(5,5)). If necessary, this additional step is repeated a total of 15 times yielding a total of 80*16 tracks for FOCSORT. If enough space is not available after filling all of the extents of all of the spill files, the FOC198 message is issued and processing terminates. **Note:** The number of extents actually obtained on any volume may be less than 16; however, in most situations 16 will be available and used.

Syntax How to Allocate a Multi-volume Data Source in the MVS Batch Environment

You have two choices for statically allocating a new multi-volume FOCUS database, Fusion database, or sequential file.

You can list multiple VOLSER identifiers on the DD card for the multi-volume data source:

```
//ddname DD DSN=dsname,VOL=SER=(vol1,…,voln),…
```

Alternatively, you can ask for multiple units of a specific type:

```
//ddname DD DSN=dsname,UNIT=(type,n),…
```

where:

ddname

Is the DDNAME associated with the multi-volume data source.

dsname

Is the data set name of the multi-volume data source.

vol1,…,voln

Are the volume identifiers for the each of the volumes to use.

type

Is the type of unit to use.

n

Is the number of units.

Allocating a Multi-volume Data Source in the TSO and MSO Environments

In both TSO and MSO you have two choices for dynamically allocating a multi-volume data source:

- You can list multiple volume identifiers.
- You can specify the number of units to use and let the system choose the specific volumes. All of the units will be the same type (for example, 3390).

Syntax How to Allocate Specific Volumes in the TSO and MSO Environments

To allocate specific volumes for a multi-volume data source, use the following syntax:

- In TSO

  ```
  TSO ALLOC ... VOLUME('vol1,...,voln')...
  ```

- In MVS FOCUS or MSO

  ```
  DYNAM ALLOC ... VOL vol1,...,voln ...
  ```

  where:

  vol1,...,voln

  Are the volume identifiers for the each of the volumes to use.

Syntax How to Specify the Number of Units in the TSO and MSO Environments

To specify the number of volumes for a multi-volume data source and let the system choose the specific volumes, use the following syntax:

- In TSO

  ```
  TSO ALLOC ... UCOUNT('n') UNIT('type') ...
  ```

- In MVS FOCUS or MSO

  ```
  DYNAM ALLOC ... UCOUNT n UNIT type ...
  ```

  where:

  n

  Is the number of volumes to use.

type

Is the type of unit to use.

Note:

- UNIT VIO is not supported.
- The RLSE option of the SPACE parameter is not supported.
- The DYNAM UCOUNT parameter is also discussed in NF670, *DYNAM Support for Unit Count*.

Example Allocating a Data Source to Two Volumes

The following DYNAM command allocates two volumes to a data source called MULTVOL:

```
DYNAM ALLOC FI MULTVOL DS USER1.FOCTST.MULTVOL TRACK SPACE 4 4 REU -
               UCOUNT 2 UNIT SYSDA CATALOG
```

With this allocation, a second volume will be used when the 17th extent is needed.

Syntax How to Display the Volume Identifiers Allocated to a Multi-volume Data Source

To see the data set information associated with a specific DDNAME, issue the following command

```
? TSO DDNAME ddname
```

where:

ddname

Is the DDNAME allocated to the data set whose volume identifiers you want to see.

Example Displaying Multi-volume Data Set Information

The following example shows how to display data set information for DDNAME MULTVOL:

```
? tso ddname multvol
```

The following information is returned. Notice that two volume serial identifiers are listed on the VOLSER line:

```
DDNAME      =  MULTVOL
DSNAME      =  USER1.FOCTST.MULTVOL
DISP        =  NEW
DEVICE      =  DISK
VOLSER      =  MFOC02,MFOC01
DSORG       =  PS
RECFM       =  F
SECONDARY   =     4
ALLOCATION  =  TRACKS
BLKSIZE     =     4096
LRECL       =     4096
TRKTOT      =       92
EXTENTSUSED =       23
BLKSPERTRK  =       12
TRKSPERCYL  =       15
CYLSPERDISK =     2227
BLKSWRITTEN =     1104
FOCUSPAGES  =     1059
>  >
```

Choosing Default Sizes for FOCUS-created Files

IBITABLA, the file that contains default allocations for FOCUS-created files, has been enhanced to allow a unit count for FOCUS databases, Fusion databases, and all sequential FOCUS-created files. The advantages of multi-volume allocations are described in *Advantages of Multi-volume Data Sources*.

Every FOCUS release is shipped with a member of the FOCCTL.DATA PDS named IBITABLA. This member contains default allocations for all FOCUS-created files that you do not specifically allocate in your FOCUS session, JCL, or CLIST. As part of the installation process, the FOCUS installer should copy the new version of IBITABLA to the ERRORS PDS and edit it to conform to the standards for the site.

When a new release of FOCUS is shipped, the installer should compare the latest version of IBITABLA to the prior site-specific version in order to construct a new site-specific version of IBITABLA. In this way, the installer will be aware of changes in format (such as new fields or ddnames added in the new release) that must be addressed in the customized copy.

The following is the default IBITABLA shipped with FOCUS Version 7.0 Release 9:

```
*...+....1....+....2....+....3....+....4....+
HOLD     CYLS   5  10                       3,
HOLDMAST TRKS   5   5 36                     ,
SAVE     CYLS   5  10                       3,
REBUILD  CYLS   5  10                       3,
FOCSML   CYLS   5   5                       2,
FOCUS    CYLS   5   5                       1,
FOCSTACK TRKS   5   5                       2,
FOCSORT  CYLS   5   5                       1,
OFFLINE  CYLS            A                   ,
SESSION  TRKS   5   5                       2,
FOCCOMP  TRKS   5   5 12                     ,
HOLDACC  TRKS   5   5 12                     ,
FMU      TRKS   5   5 12                     ,
TRF      TRKS   5   5 12                     ,
FOCPOOLT CYLS   5  20             NOHIPER   4,
FUSION   CYLS   5  50             NOHIPER   4,
MDI      CYLS   5  20             NOHIPER   4,
FOC$HOLD CYLS   5   5                       2,
EXTINDEX CYLS   5   5                       2,
```

The unit count field is columns 44-45. The fields are:

Columns	Length	Comments
01-08	8	Class of file - DDname
10-13	4	Allocation units (CYLS, TRKS)
15-17	3	Primary allocation

Columns	Length	Comments
19-21	3	Secondary allocation
23-24	2	Number of directory blocks (blank specifies a sequential file; 0 is invalid)
26-26	1	SYSOUT class
28-33	6	Volume serial on which to allocate
35-42	8	Type of unit to allocate (for example, 3390, DASD, NOHIPER*)
44-45	2	Unit count

Note that partitioned data sets do not support multi-volume allocations.

If you enter a non-valid value for any field in IBITABLA, FOCUS substitutes the corresponding value from the original version of IBITABLA that was shipped with FOCUS.

Example Allocating DDNAME FOCSTACK to Two Volumes in IBITABLA

Assume you replaced the original FOCSTACK line with the following in IBITABLA:

```
|...+....1....+....2....+....3....+....4....+
FOCSTACK TRKS   5   5             3390      2,
```

This line indicates the following allocation attributes:

- DDname = FOCSTACK
- Allocation is in tracks:

  Primary tracks=5

  Secondary tracks=5
- Type of unit = 3390
- Number of units=2

The resulting dynamic allocation is equivalent to the following JCL:

```
//FOCSTACK DD SPACE=(TRK,(5,5)),UNIT=(3390,2),...
```

Reference Usage Notes for IBITABLA

- IBITABLA is a fixed-format file.
- Each data field must be placed in specific columns, but leading or trailing blanks are allowed. (There is at least one blank between successive data fields.)
- All lines beginning with an asterisk (*) are comments.
- Unit count is ignored for HiperFOCUS files with DISP=(NEW,DELETE).

NF584: Dynamically Setting the IDMS DBNAME and DICTNAME

The new IDMS Interface commands SET DBNAME and SET DICTNAME enable you to dynamically change the DBNAME and DICTNAME parameters at any time during your FOCUS session.

If you do not issue these commands, the Interface reads the DBNAME and DICTNAME from the Access File. However, once you issue them, the new DBNAME and DICTNAME values take precedence over those in your Access Files. The new values remain in effect until you either:

- Reissue the SET commands with new DBNAME and DICTNAME values.
- End your FOCUS session.
- Reinstate the Access File parameters by issuing the SET commands with the DEFAULT option.

Syntax How to Set the DBNAME and DICTNAME

Issue the following commands at the FOCUS session prompt, in a FOCEXEC, or in any supported profile:

```
TSO IDMSR SET DBNAME {dbname|DEFAULT}
TSO IDMSR SET DICTNAME {dictname|DEFAULT}
```

where:

dbname

Is the IDMS database name that you want to access.

dictname

Is the IDMS dictionary name that you want to access.

DEFAULT

Causes the Interface to read the value from the Access File.

Syntax How to Display the Current Settings

To display the settings that are currently in effect, issue the following command:

```
TSO IDMSR SET ?
```

Example Dynamically Changing the DBNAME and DICTNAME Values

```
TSO IDMSR SET DBNAME EMPDEMO
TSO IDMSR SET DICTNAME APPLDICT
```

NF673: Model 204 Interface Account Split

The Model 204 Interface has been enhanced to accept an account code as part of the logon string.

The syntax for specifying the logon account in the Access File is

```
ACCOUNT=x[ y], ACCOUNTPASS=pswd, IFAMCHNL=ifchnl,$
```

The SET command syntax for specifying the logon account is

```
{TSO|MVS} M204IN SET ACCTNAME x[ y]
{TSO|MVS} M204IN SET ACCTPASS pswd
```

where:

`TSO|MVS`

Is a required environmental prefix. TSO and MVS are synonyms and can be used interchangeably.

x

Is the user ID, up to 16 characters.

y

Is the account code, up to 15 characters, preceded by exactly one blank.

pswd

Is the account password.

Example Setting the Model 204 Userid and Account Code

Assume the Model 204 userid is SUPERKLUGE and the account code is MIS.

You can issue the following Interface command to set these values:

```
TSO M204IN SET ACCTNAME SUPERKLUGE MIS
```

You can also set them in the Access File with the following attribute:

```
ACCOUNT=SUPERKLUGE MIS, ...
```

NF720: SQLJOIN OUTER Setting for Relational Interfaces

With the new SQLJOIN OUTER setting you can control when the relational Interfaces optimize outer joins without affecting the optimization of other operations. This parameter provides backward compatibility with prior releases of the relational Interfaces and enables you to fine-tune your applications.

When join optimization is in effect, the Interface generates one SQL SELECT statement that includes every table involved in the join. The RDBMS can then process the join. When join optimization is disabled, the Interface generates a separate SQL SELECT statement for each table, and FOCUS processes the join.

In FOCUS Version 7.0 Release 6, the relational Interfaces were enhanced to optimize outer joins. The following command causes the Interface to generate an outer join:

```
SET ALL = ON
```

Beginning with FOCUS Version 7.0 Release 6, issuing this command with OPTIMIZATION enabled invokes outer join optimization. In FOCUS releases prior to 7.0.6, this setting disabled optimization so that FOCUS always processed the outer join. Turning OPTIMIZATION OFF still causes FOCUS to process the join, but it disables *all* optimization enhancements, not just outer join processing.

Starting with FOCUS Version 7.0 Release 9, you can use the SQLJOIN OUTER setting to disable outer join optimization while leaving other optimization enhancements in effect.

Syntax How to Control Outer Join Optimization

Issue the following command at the FOCUS prompt, in a stored procedure, or in any supported profile:

```
SQL target_db SET SQLJOIN OUTER {ON|OFF}
```

where:

target_db

Indicates the target RDBMS. Valid values are DB2, SQLDS, SQLDBC, SQLORA, or SQLIDMS. Omit if you issued the SET SQLENGINE command.

ON

Enables outer join optimization.

OFF

Disables outer join optimization. OFF is the default value.

Note:

- The SQLJOIN OUTER setting is available only when optimization is enabled (that is, OPTIMIZATION is not set to OFF).
- The SQLJOIN OUTER setting is ignored when SET ALL = OFF.

Effects of Combinations of Settings on Outer Join Optimization

The following table describes how different combinations of OPTIMIZATION and SQLJOIN OUTER settings affect Interface behavior. It assumes that SET ALL = ON:

Settings		**Results**	
OPTIMIZATION	**SQLJOIN OUTER**	**Outer Join Optimized?**	**Other Optimization Features**
ON	ON	Yes	Enabled
ON	OFF	No	Enabled
OFF	N/A	No	Disabled
SQL	ON	Yes, in all possible cases	Enabled
SQL	OFF	No	Enabled
FOCUS	ON	Yes if results are equivalent to FOCUS managed request	Enabled
FOCUS	OFF	No	Enabled

Reference SQLJOIN OUTER Messages

If SQLJOIN OUTER is set to OFF, the following message displays when you issue the SQL ? query command:

```
(FOC1420) OPTIMIZATION OF ALL=ON AS LEFT JOIN -  : OFF
```

NF652: Teradata Interface Kanji Support

The Teradata Interface now supports the DBCS (Double Byte Character Set) Kanji characters described by the Teradata datatypes GRAPHIC and VARGRAPHIC.

The following conversion chart shows the SQL datatypes that support the Kanji character set and their corresponding ACTUAL formats in the Master File:

SQL Datatype	ACTUAL Format	Description
GRAPHIC	Kn	DBCS Kanji character set. Fixed-length string of 'n' 16-bit characters where $0 < n \leq 127$. The appropriate corresponding USAGE format is A(2n+2).
VARGRAPHIC	Kn	DBCS Kanji character set. Varying-length string of 'n' 16-bit characters where $0 < n \leq 127$. The appropriate corresponding USAGE format is A(2n+2). **Note:** LONG VARGRAPHIC (or VARGRAPHIC (n) where n>127) is not supported.

NF722: FOCUS Client DNS Names Support

This feature enables the FOCUS Client EDACFG file to identify an EDA server by host name rather than IP address. The EDACFG file is the client communications configuration file allocated to DDNAME EDACFG. In prior versions of FOCUS Client, the EDACFG file had to supply the IP address of the EDA server to which it would connect.

The domain name system (DNS) is a global network of servers that translate host names, such as www.ibi.com, into IP addresses. For more information about FOCUS Client, see NF494, *FOCUS Client - Remote Data Access via EDA/SQL*, which was implemented in FOCUS 7.0.5.

Syntax How to Invoke DNS Names Support

The syntax for specifying a host name in the client configuration file for TCP/IP is

```
HOST = hostname
```

where:

hostname

Is the host name of the EDA server.

Example Using DNS Names Support

The following client configuration file is used for connecting to the host named IBIMVS:

```
NAME = EDA/SQL CLIENT USING CS/3 TCP/IP
NODE = TCPOUT
  BEGIN
 ;  TRACE = 31
    PROTOCOL = TCP
    CLASS = CLIENT
    HOST   = IBIMVS           ; DNS NAME OF HOST
    SERVICE   = 2459          ; PORT NUMBER OF THE EDA SERVER
  END
```

NF656: Controlling REBUILD Messages

This feature allows for direct control over the frequency with which REBUILD issues messages. FOCUS, by default, displays a message for every 1000 records read during the database retrieval and load phases of the REBUILD utility. The message, REFERENCE..AT SEGMENT 1000, REFERENCE..AT SEGMENT 2000.. is a function of the number of records in the FOCUS file being rebuilt. The frequency of these messages can become problematic for larger FOCUS files because CMS spool space may be limited.

Syntax How to Control REBUILD Messages

The user can set how often the message is displayed by issuing the command:

```
SET REBUILDMSG = n
```

where:

n

is any 8-byte integer.

A setting of less than 1000 generates a diagnostic and keeps the current setting. The current setting will either be the default of 1000, or the last valid integer greater than 999 to which REBUILDMSG was set. A setting of 0 disables the ‘REFERENCE..AT SEGMENT’ messages.

Example Controlling Display of REBUILD Messages

The following example shows a REBUILD CHECK function where REBUILDMSG has been set to 4000, and the database contains 19,753 records.

```
ENTER NAME OF FOCUS/FUSION FILE (FN FT FM)
...
STARTING..
REFERENCE..AT SEGMENT    4000
REFERENCE..AT SEGMENT    8000
REFERENCE..AT SEGMENT   12000
REFERENCE..AT SEGMENT   16000
NUMBER OF SEGMENTS RETRIEVED=   19753
CHECK COMPLETED...
```

NF670: DYNAM Support for Unit Count

The DYNAM command now supports the unit count parameter for allocating multi-volume data sources. With the UCOUNT parameter you can allocate a file to multiple volumes of a particular type of storage device without indicating specific volume serial numbers.

Advantages of Multi-volume Data Sources

Many sites prefer to distribute high volume data sources across multiple volumes in order to manage:

- Use of storage on specific devices or device types.
- Run-time access to these devices.

This new DYNAM parameter is particularly useful for allocating large temporary files such as FOCSORT, FOCPOOLT, and HOLD files in the TSO and MSO environments.

Syntax How to Specify the DYNAM Unit Count Parameter

```
DYNAM ALLOC ... UCOUNT n UNIT type ...
```

where:

n

Is the number of volumes to use.

type

Is the type of unit to use. **Note:** UNIT VIO is not supported.

Example Allocating a HOLD File to Two Volumes

The following DYNAM command allocates two 3390s to a HOLD file:

```
DYNAM ALLOC FILE HOLD SPACE 20,10 TRACKS UCOUNT 2 UNIT 3390
```

NF691: Escape Character for LIKE Predicate

New syntax exists for the LIKE predicate that enables you to search for special characters in data. You can set an escape character to use in the LIKE predicate. Then, when you include this escape character in front of a special character in the mask pattern, the parser treats the special character as a normal character and searches for it in the data. This next character can be '%' or '_' which are normally special characters. LIKE may be coded in WHERE, DEFINE, and COMPUTE statements. The mask is an alphanumeric, user-supplied pattern that FOCUS uses to compare characters in a data field value. A mask has two special characters:

1. The percent sign (%) to indicate any number of characters; and
2. The underscore (_) to indicate a single character in a specified position.

The escape character enhancement permits FOCUS to treat these masking characters as literals within the search pattern and not as wildcards.

Any single character can be used as an escape character, but the one used must be prefaced with the word ESCAPE. The syntax is:

```
WHERE field LIKE 'phrase' ESCAPE 'c'
```

where

`'c'`

Is any character embedded in the phrase before a '%' or '_'.

Escape Character Capabilities

The escape character is supported by the SQL Interfaces, as well as FOCUS databases. The escape character is only in effect when the ESCAPE syntax is included in the LIKE predicate.

Syntax How to Use the Escape Character

The pattern (alphanumeric constant) that follows the LIKE predicate may contain wildcard characters ('%') and ('_') that users may need to escape in order to use as part of the search pattern. Every LIKE predicate can provide its own escape character to be used within the pattern (or mask). The syntax is as follows:

```
WHERE expression LIKE 'abc\%' ESCAPE '\'
```

where

expression

Is the name of the expression to be evaluated in the selection test.

'abc\%'

Is the alphanumeric test value.

'\'

Is the escape character enclosed in single quotation marks. When the escape character is used in the pattern immediately preceding the special character '%' or '_', FOCUS is instructed to treat the special character as a literal and not as a wildcard. The character itself can also be escaped, thus becoming a normal character in a string (for example, `'abc\%\\'`).

Example Using the Escape Character

Using the Car file, assume that both Peugeot and Alfa Romeo produce 4_door car models. To generate a report showing countries that produce 4_door car models, the syntax would be as follows:

```
TABLE FILE CAR
PRINT CAR MODEL
BY COUNTRY
WHERE MODEL LIKE '%g_DOOR%' ESCAPE 'g'
END
```

The request produces the following report:

```
PAGE     1

COUNTRY       CAR             MODEL
-------       ---             -----
FRANCE        PEUGEOT         504 4_DOOR
ITALY         ALFA ROMEO      2000 4_DOOR BERLINA
```

Reference Special Considerations

- The use of an escape character in front of any character other that ‘%’, ‘_’, and itself will be ignored.
- Only one escape character can be used per LIKE phrase.
- If a WHERE clause is used with lazy ORs, the ESCAPE must be on the first phrase and will apply to all subsequent phrases in that WHERE clause. For example:

  ```
  WHERE field LIKE 'ABCg_' ESCAPE 'g' OR 'ABCg%' OR 'g%ABC'
  ```

Reference Error Messages

(FOC36251) SYNTAX ERROR IN LIKE OPERATOR

Alphanumeric mask must follow the LIKE operator. An optional keyword ESCAPE followed by a single character alphanumeric constant in apostrophes can be used after the mask.

NF718: DYNAM Support for Existing Relative GDG Numbers

The DYNAM command now supports allocation of existing iterations of a Generation Data Group (GDG) using relative index numbers. Existing iterations are those with index numbers less than or equal to zero.

Note: DYNAM does *not* support allocation of a new GDG iteration; that is, you cannot use the relative number +1 in a DYNAM allocation for a GDG.

Using DYNAM With Relative GDG Numbers

The original definition of a GDG assigns it a group name and specifies how many generations will be maintained. Once the maximum number of generations has been reached, each new iteration replaces the oldest existing iteration. The data set name for each iteration is the group name appended with a qualifier that contains the iteration number.

For example, the following GDG, named USER1.HOLD.GDG, has three generations:

Data Set Name	Iteration Number	Index Number
USER1.HOLD.GDG.G0003V00	3 (Oldest)	-2
USER1.HOLD.GDG.G0004V00	4	-1
USER1.HOLD.GDG.G0005V00	5 (Current)	0

The current (that is, newest) iteration always has the highest iteration number and corresponds to index number zero. The next newest corresponds to index number -1, and so on.

Prior to this new feature, DYNAM could allocate only the current iteration of a GDG. Now DYNAM supports relative index numbers for allocating any existing iteration.

Syntax How to Use DYNAM With Relative Index Numbers

Use the following syntax to allocate an existing iteration of a GDG

```
DYNAM ALLOC FILE ddname DS 'group_name(index)' SHR REUSE
```

where:

ddname

Is the name of the Master File for the GDG.

group_name

Is the group name of the GDG.

index

Is the index number for an existing iteration of the GDG. It must be less than or equal to zero.

Example Allocating Existing GDG Iterations Using DYNAM With Relative Index Numbers

The GDG named USER1.HOLD.GDG discussed in *Using DYNAM With Relative GDG Numbers*, is described by Master File GDG1:

```
FILENAME=GDG1,SUFFIX=FIX
SEGNAME=ORIGIN,SEGTYPE=S1
 FIELDNAME=FLD1,,A10,$
 FIELDNAME=FLD2,,A20,$
 FIELDNAME=FLD3,,A30,$
 FIELDNAME=FLD4,,A10,$
```

In each iteration, the value stored in FLD1 is the generation number. For example, USER1.HOLD.GDG.G0004V00 contains the following records:

```
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

The following request allocates and prints FLD1 of iteration four (index number -1)

```
> dynam alloc file gdg1 ds 'user1.hold.gdg(-1)' shr reu
>  > table file gdg1
> print fld1
> end
```

```
PAGE     1


FLD1
----
4444444444
4444444444
4444444444
4444444444
4444444444
```

The following DYNAM command allocates the current iteration of the GDG, and the TABLE request accesses this latest iteration:

```
> dynam free file gdg1
>  > dynam alloc file gdg1 ds 'user1.hold.gdg(0)' shr reu
>  > table file gdg1
> print fld1
> where recordlimit eq 1
> end
PAGE     1


FLD1
----
5555555555
```

Syntax Determining Which GDG Iteration is Allocated

To determine which GDG iteration is allocated, issue the ? TSO DDNAME query command:

```
>  > ? tso ddname gdg1
```

```
DDNAME       =  GDG1
DSNAME       =  USER1.HOLD.GDG.G0003V00
DISP         =  SHR
DEVICE       =  DISK
VOLSER       =  USERME
DSORG        =  PS
RECFM        =  FB
SECONDARY    =     5
ALLOCATION   =  TRACKS
BLKSIZE      =      8000
LRECL        =        80
TRKTOT       =         1
EXTENTSUSED  =         1
BLKSPERTRK   =         6
TRKSPERCYL   =        15
CYLSPERDISK  =      3340
BLKSWRITTEN  =         1
>  >
```

The DSNAME, USER1.HOLD.GDG.G0003V00, indicates that the third iteration is allocated.

Reference Error Messages

The MESSAGE parameter must be set ON in order to receive these messages.

An attempt to allocate a new iteration of a GDG (index number +1) generates the following error message:

```
(FOC880)       DYNAMIC ALLOCATION ERROR: IKJ56871I DATA SET name
            NOT ALLOCATED, RELATIVE GENERATION NUMBER INCOMPATIBLE FOR
            SPECIFIED STATUS, #099:0C-0394
```

Allocating an index number that is out of range (too large a negative number), generates the following message:

```
(FOC855)       UNABLE TO LOCATE DATASET name, #099:04-1708
```

NF735: Enhancement to ? SET

Two options have been added to the ? SET command. They are ? SET FOR and ? SET NOT. The FOR option lists the current state of the command queried, and details where it may be set within FOCUS. The NOT option produces a list of SET commands not settable in five specific areas.

Syntax Querying a Command

```
? SET FOR parameter
```

where:

parameter

Is any SET parameter.

Example Querying the Current State of EXTSORT

Entering

```
? SET FOR EXTSORT
```

yields

```
EXTSORT                    ON

 ------------------------------------------------
 SETTABLE FROM COMMAND LINE                   :  YES
 SETTABLE ON TABLE                            :  YES
 SETTABLE FROM SYSTEM-WIDE PROFILE            :  YES
 SETTABLE FROM HLI PROFILE                    :  YES
 POOL TABLE BOUNDARY                          :  YES
>
```

The preceding screen shows that EXTSORT is currently set ON and that it is settable from all five features.

Syntax Determining Which Commands Are Not Settable In Each of the Five Features

```
? SET NOT functional_area
```

where:

functional_area

can be one of the following five areas:

- PROMPT (in a PROMPT command)
- ONTABLE (in a report request)
- FOCPARM (in the FOCPARM profile)
- HLIPROF (in the HLI profile)
- PT (Pooled Tables)

Example Determining Which Commands Are Not Settable In a Report Request

Entering

```
? SET NOT ONTABLE
```

yields:

```
                    NON-SETTABLE ON TABLE PARAMETER SETTINGS

 BINS              64    LANGUAGE      AMENGLISH    REBUILDMSG        1000
 BLKCALC          NEW    MAXPOOLMEM        32768    SAVEMATRIX          ON
 BYPANELING       OFF    MDIBINS            8000    TCPIPINT           OFF
 CACHE              0    MDIPROGRESS      100000    TEMP DISK            A
 COLUMNSCROLL     OFF    MODE                CMS    TRMSD               24
 DATEDISPLAY      OFF    MPRINT              NEW    TRMSW               80
 DATEFNS           ON    POOL                OFF    TRMTYP        1 (3270)
 DEFCENT           19    POOLBATCH           OFF    WEBHOME            OFF
 EUROFILE                POOLFEATURE         OFF    WIDTH              130
 FIELDNAME        NEW    POOLMEMORY        16384    WINPFKEY           OLD
 FOCSTACK SIZE      8    POOLRESERVE        1024    YRTHRESH             0
 HTMLMODE         OFF    PRINTPLUS          OFF
>
```

The preceding screen shows a list of parameters that are not settable using ON TABLE.

NF740: Changes to the REBUILD Prompt

With the addition of two new options, MIGRATE and MDINDEX, the REBUILD prompt no longer fits on one line. After you enter REBUILD at the FOCUS prompt a numbered list is displayed that allows you to enter either the option name or its corresponding number.

Syntax The REBUILD Prompt Screen

```
>REBUILD
Enter option
1. REBUILD           (Optimize the database structure)
2. REORG             (Alter the database structure)
3. INDEX             (Build/modify the database index)
4. EXTERNAL INDEX     (Build/modify an external index database)
5. CHECK             (Check the database structure)
6. TIMESTAMP          (Change the database timestamp)
7. DATE NEW          (Convert old date formats to smartdate formats)
8. MDINDEX             (Build/modify a multidimensional index)
9. MIGRATE             (Convert FOCUS masters/DBs to FUSION)
```

Note: A change was also made to the FILENAME prompt. Instead of prompting for the name of the FOCUS file, you are now asked for the FOCUS/FUSION filename. Within the individual options a similar numbered list is also displayed if applicable. REBUILD commands are stackable in name or number format.

NF745: ? PTF Enhancements

Two columns have been added to the output displayed in response to ? PTF. They are, SUPERSEDED BY, which displays a superseding PTF if one exists and PUT LEVEL, which demonstrates the order in which multiple PTFs must be applied.

Syntax How to Identify PTFs Applied to Your Version of FOCUS

Issue the following command at the FOCUS prompt:

```
? PTF
```

A screen similar to the following is displayed:

```
>
? ptf
PTFS APPLIED TO RELEASE 70XFOC
FROM PTFTABLE LOCATED IN IBITEST LOADLIB C1

 COUNT PTF NUM   CREATED   APPLIED   SUPERSEDED BY   PUT LEVEL
 ----- -------   -------   -------   -------------   ---------
    1) 95828     ........  ........  112600          ......
    2) 107164    ........  ........  112600          ......
    3) 110763    ........  ........  112600          ......
    4) 112600    19990427  19990513  ......          200295
>
```

Note: Dots denote that no information exists for a column entry in the resulting report.

NF746: Leading Zeros

This feature may be used in Dialogue Manager for date subroutines that return a numeric integer format. It specifically addresses the case where a two-digit year is input in a Dialogue Manager string. The result of the subroutine is 00, representing the year 2000. The leading zeros are truncated when typed out.

Syntax Displaying Leading Zeros

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

where:

ON

Allows the display of leading zeros if they are present.

OFF

Truncates leading zeros if they are present.

Example Preserving Leading Zeros

In the following example the AYM subroutine is being called. The input year is 99 and the month is 12.

```
-SET &IN = '9912';
-SET &OUT = AYM (&IN, 1, 'I4');
-TYPE &OUT
```

This yields

```
1
```

Adding

```
SET LEADZERO = ON
```

before the above example yields

```
0001
```

correctly indicating January. 2000.

Note: LEADZERO only supports expressions that make a direct call to a subroutine. Expressions that have nesting or other mathematical functions truncate leading zeros. For example,

```
-SET &OUT = AYM(&IN, 1, 'I4'/100;
```

NF748: HOLD FORMAT WP With Carriage Control

You can now control whether carriage control characters appear in column 1 of each page of report output saved with the HOLD FORMAT WP option. When you choose to include carriage control in a format WP file in MVS, FOCUS creates the file with RECFM VBA so that each report page prints on a separate sheet of paper.

Syntax How to Include Carriage Control Characters in a FORMAT WP Hold File

```
[ON TABLE] HOLD AS filename FORMAT WP [CC|NOCC]
```

where:

filename

Is the name of the resulting file.

CC

Includes a carriage control character (1) as the first character of each report page in the extract file. In MVS, the file is created with RECFM VBA which causes the operating system to respect the carriage control via page ejects.

NOCC

Omits carriage control characters from the extract file. The file is created with RECFM VB.

When you do not include either CC or NOCC in the syntax, the value of the PAGE-NUM (or PAGE) parameter controls whether carriage control characters are supplied in column 1:

- SET PAGE-NUM=ON (the default) or SET PAGE-NUM=NOPAGE causes column 1 to be blank (no carriage control). In MVS, the file is created with RECFM=VB.
- SET PAGE-NUM=OFF, SET PAGE-NUM=TOP, or using the TABPAGENO option in the heading of the report request supplies carriage control characters. This is the same as prior behavior. However, starting with this release, these settings cause the Hold file to be created with RECFM VBA in MVS.

Reference Directing Output to a Printer

In MVS, you can send the format WP output directly to a printer by allocating the filename to that printer. You must FREE or CLOSE the file after it is created and before it is printed. For example:

```
SET PAGE = ON
  DYNAM ALLOC FILE EMPLOYEE DA PREFIX.EMPLOYEE.FOCUS SHR REU
  DYNAM FREE FILE HOLD1
  DYNAM ALLOC FILE HOLD1 SYSOUT A DEST IBIVM.P29C1 CLOSE REUSE
  TABLE FILE EMPLOYEE
  PRINT EMP_ID
  BY CURR_SAL BY LAST_NAME
   ON TABLE HOLD AS HOLD1 FORMAT WP CC
END
```

Note: Your SYSOUT allocation must contain your destination printer.

In VM/CMS, print the HOLD file using the CC option of the CMS PRINT command to interpret the first character of each record as a carriage control character.

```
PRINT fn ft fm (cc
```

Reminder: If the HOLD file is created with carriage control, it must be printed with carriage control.

Reference FORMAT WP Carriage Control and MVS Record Format Options

The following table summarizes the options for creating a FORMAT WP extract file:

Carriage Control Option	PAGE=ON or NOPAGE		PAGE=OFF or TOP or TABPAGENO in heading	
	Carriage Control?	MVS RECFM	Carriage Control?	MVS RECFM
CC	Yes	VBA	Yes	VBA
NOCC	No	VB	No	VB
none	No	VB	Yes	VBA

Index

Symbols

$ VIRT
&FOCERRNUM in Adabas Write
&FOCNEXTPAGE
&FOCUSER
&LOC_LIST
&RETCODE
? FUNCTION command
? MDI query command
? MDIPROGRESS query command
? PTF
? SET FOR
? SET NOT
? STYLE command

Numerics

1024 display objects
16K page
495 display objects
4K alpha

A

absolute starting positions

Access File for Adabas Write
  sample
Access File for FOCUS data sources
  attributes
  creating
  DATANAME attribute
  introduction
  LOCATION attribute
  MASTERNAME attribute
  syntax
  WHERE attribute
ACCESSFILE attribute
ACCTNAME
ACCTPASS
ACROSS attribute
ACROSS field
ACROSS limit
ACROSS values
ACROSSCOLUMN attribute
  WHEN
ACROSSTITLE component
ACROSS-TOTAL
ACROSSVALUE component
ACTUAL format for date-time data type
  relational data source

Adabas Write Data Adapter
  Access File changes
  delete rules
  error messages
  examples
  INSERT and UNQKEYNAME
  insert rules
  Master File changes
  non-updatable fields in MAINTAIN
  non-updatable fields in MODIFY
  options
  return codes
  sample Access File
  sample Master File
  synonyms
  testing FOCERROR
  transaction control
  UPDATE rules
  user errors
ADD
adding graphics to a report
aggregating
  and sorting report columns
  controlling retrieval order
  using External Sort
aligning decimal points
aligning heading items
aligning headings and footings

aligning text fields
aligning using HTML tables
alignment methods
ALL parameter
  and MULTIPATH
ALLOC for multiple volumes
allocating
  long Master File names
  multiple units
  multi-volume data sources
alphanumeric field on summary line
alphanumeric fields, long
ALT attribute
amper variables
  &FOCNEXTPAGE
  &FOCUSER
  long
  TABLASTPAGE
answer set, reducing size of
AnV format
applying CSS styles
applying grids to a report
arguments, verifying in functions
assignment of date-time values
attribute inheritance
  augmenting attributes
  overriding
attributes for FOCUS Access Files

AUTODATE
AUTODB2
  and DDF
  batch execution
  date-time support
AUTOIMS
  accessing Master File and FOCPSB
  CLIST
  COBOL FD translator window
  default parameters
  installing
  invoking
  Master File
  PF keys
  prerequisites
  PSB name
  required data sets
AUTOINDEX
AUTOSQL
  date-time support

B

background images
BACKIMAGE attribute
BASEURL SET parameter
batch
  allocating multi-volume data sources
  error processing

batch error processing
blank lines
  formatting
blank spaces
  above data values
  adding around a report
BLANKINDENT
BORDER attribute
borders
  around headings and footings
  colors
  styles
  width of
bottom margins
BOTTOMGAP attribute
BOTTOMMARGIN attribute
British pound symbol
browser fonts
browser support for Cascading Style Sheets
browser titles
buffers for XFOCUS data sources
BY attribute
BY TOTAL
BYDISPLAY

C

CAF
  SET SSID

CA-IDMS Data Adapter
  NF584 Dynamically Setting the IDMS DBNAME and DICTNAME
calculated value, sorting by
calling subroutines written in REXX
CALLTYPE
  with NEXT
CAPTION
captions
carriage control in FORMAT WP files
Cascading Style Sheet classes
  assigning to components
  assigning to report components
  naming
Cascading Style Sheet rules
Cascading Style Sheets (CSS)
  browser support
  classes
  conditional styling
  external
  formatting
  hyperlinks
  images
  inheritance
  internal
  linking to
  naming classes
  report formatting
  rules
  troubleshooting

case logic
CC
CDN
CENT-ZERO
changes to the REBUILD prompt
changing retrieval order with aggregation
CHECK STYLE
CICS
  MSO control blocks
CLASS attribute
classes in Cascading Style Sheets
  assigning to components
  assigning to report components
clearing DEFINE functions
CMS
  FILEDEF for creating extract files
COLOR attribute
color values
colors
  page
COLSPAN attribute
COLUMN attribute
column titles, raised limit

columns
  controlling order
  controlling spacing
  determining width
  identifying in a style sheet
  justifying
  positioning
  stacking
  titles of
COM output file format
COM suffix
COMBINE
  command limit
  diagram
  different SUFFIX
  FIND
  versus JOIN
combining CSS with other formatting methods
COMMA output file format
comma suppress edit format option
comma-delimited files
comments in StyleSheets
COMMIT WORK
comparison of date-time values
compile REXX subroutines in CMS
compiled DEFINEs
components for date-time functions
compound PDF and PostScript reports

COMPUTE
  and currency conversion
  and date-time values
  sorting by
COMT output file format
COMT suffix
conditional grid formatting
conditional join
conditional styling
  applying to FORECAST fields
  sequential conditional formatting
  StyleSheets and
  WHEN attribute
configuring
CONTAINS
continental decimal notation
control blocks for MSO/CICS
controlling REBUILD messages
controlling report page numbering
creating comma-delimited files

CSS (Cascading Style Sheets)
  browser support
  conditional styling
  external
  formatting
  hyperlinks
  images
  inheritance
  internal
  linking to
  naming classes
  report formatting
  troubleshooting
CSSURL attribute
CSSURL parameter
cumulative standard normal distribution
CURR
currency
  conversion
  conversion calculations
  conversion database
  conversion processing
  euro support
  field in a data source
  sample codes
  symbols
CURRENCY_ID
custom report titles

custom reports
custom worksheet names

D

data
Data Adapter Enhancements
  Adabas Write Data Adapter
  AUTOIMS
  DB2 Data Adapter SET SSID Command for MSO
  DB2 FETCH FIRST n ROWS Support
  Dynamically Setting the IDMS DBNAME and DICTNAME
  FIELDTYPE=R
  Increased Row Size Limit for Relational Data Adapters
  Model 204 Interface Account Split
  Optimizing Selection of Relational Variable Length Character Data Types
  Oracle Data Adapter IXSPACE Setting
  SQL Support for Date-Time Formats
  SQLJOIN OUTER Setting for Relational Interfaces
  Teradata Interface Kanji Support
DATA component
data source, token delimited
data type, date-time
  ACTUAL format
  and SQL Translator
  entering values
  for relational data sources
  USAGE

Database Enhancements
  DATASET for a Segment
  File Extensions for Maintain Procedures
  FOCUS SETs From Maintain
  GROUP Fields in FOCUS Data Sources
  Long Index Names
  Long Segment Names
  MDI for FOCUS and XFOCUS Data Sources
  Screening Conditions for Maintain
  XFOCUS Data Source
database page, large
DATANAME
DATANAME attribute
DATASET
  behavior in FOCUS data sources
  priority in the Master File
  syntax for FOCUS data sources
DATASET attribute
  for a cross-referenced segment
  for a LOCATION segment
DATASET for a segment
DATASET in a Master File
  behavior in fixed-format sequential data sources
  behavior in FOCUS data sources
  behavior in VSAM data sources
  priority in the Master File
  syntax for fixed-format data sources
  syntax for FOCUS data sources
  syntax for VSAM data sources

DATE format
date string
  formatted string format
  numeric string format
  translated string format
DATEFORMAT parameter
date-time data type
  ACTUAL format
  ACTUAL format for relational data source
  and HOLD files
  and missing values
  and SAVE files
  assignment
  comparison
  component names in functions
  DATEFORMAT parameter
  describing
  describing relational
  DTSTRICT parameter
  entering values
  formatted string date format
  functions
  HADD function
  HCNVRT function
  HDATE function
  HDIFF function
  HDTTM function
  HGETC function

date-time data type (continued)
  HINPUT function
  HMIDNT function
  HNAME function
  HPART function
  HSETPT function
  HTIME function
  ISO standards
  numeric string date format
  optimization for relational data sources
  SQL Translator support
  translated string date format
  USAGE
  WEEKFIRST parameter
  with AUTODB2/SQL
date-time formats
  and SQL
DB2 Data Adapter
  FETCH FIRST n ROWS support
  SET SSID Command for MSO
  TIMESTAMP support
DB2 data sources
  and date-time data type
  and long Master File names
DBCS character support in Teradata
DBNAME
DCB attributes
  for MDI
  for XFOCUS data sources

DDF
DDNAME
  for Maintain procedures
  For REXX subroutines
decimal alignment of headings
decimal points
  aligning
decimal values
  comparing
declarations
DEFAULT- FIXED attribute
default formatting
DEFAULT- PROPORTIONAL attribute
default proportional fonts
DEFINE
  and currency conversion
  and date-time values
  compiled
  preserving fields
DEFINE FILE SAVE and DEFINE FILE RETURN
DEFINE functions
  clearing
  querying
DELETE
  rules for Adabas data sources
delimiter
descriptors in Adabas Write
Dialogue Manager TRUNCATE Function

Dialogue Manager variables
  &FOCNEXTPAGE
  &FOCUSER
  long
  TABLASTPAGE
DICTNAME
DIMENSION
dimensional joins
  choosing dimensions
  guidelines
display fields
  raised limits
display formats for reports
  Excel 2000
  Excel 97
  HTML
  PDF
  PostScript
  PS
display objects, 1024
displaying fractions
displaying reports
  in a browser
displaying WHERE-based JOINs
Distributed Data Facility
DNS names support
dollar sign
DOUBLEXP

DT format for date-time constants
DTSTRICT parameter
dummy segment (SYSTEM99)
DYNAM
  allocating multiple volumes
  for long Master File names
  support for existing relative GDG numbers
  support for new GDG iterations
  support for unit count
DYNAM ALLOC LONGNAME
DYNAM FREE LONGNAME
dynamic reformatting

E

EDACFG
ELEMENT attribute
embedded fields
embedding images
embedding text fields in headings
encoding indexed values
enhanced screening conditions for Maintain
Enhancement to ? SET (FOR, NOT)
EQ_MASK
error handling
  Adabas
  batch
  ROLLBACK WORK
  testing with &RETCODE

error messages for Adabas Write
ERROROUT parameter
ERRORRUN
errors, user
  for Adabas Write
escape character for LIKE
euro support
  calculations
  currency conversion
  currency database
  currency-denominated field
  euro symbol
  sample currency codes
EUROFILE
examples for Adabas Write
Excel (HOLD format)
Excel 2000 format
  alignment
Excel 97 format
EXECLOAD
Exit on Error
EXL2K alignment
EXPAVE
exponential moving average
exponential smoothing
expressions, increased level of nesting
EXTAGGR
extended currency symbol support

extensions
  for Maintain procedures
  to FORECAST
external Cascading Style Sheet classes
  assigning to report components
  names
external Cascading Style Sheet rules
  BODY element
  TD element
external Cascading Style Sheets
  browser support
  combining with other formatting methods
  conditional styling
  hyperlinks
  images
  inheritance
  linking to
  refreshing
  report formatting
  troubleshooting
external index, increased number of partitions
external sort
  aggregation by
  controlling retrieval order
  HOLD from
EXTHOLD

F

FETCH FIRST n ROWS
field names, qualified
field-based reformatting
FIELDTYPE=R
file extensions for Maintain procedures
FILEDEF for extract files
files, token delimited
FILETYPE for REXX subroutines
Financial Modeling Language (FML)
  FORMULTIPLE
  free text in
  hierarchies
  indenting rows
FIND
FML (Financial Modeling Language)
  FORMULTIPLE
  free text in
  hierarchies
  indenting reports

FML hierarchies
- **ADD**
- **CAPTION**
- **consolidating**
- **displaying**
- **GET CHILDREN**
- **GET CHILDREN ADD**
- **LOAD CHART**
- **loading manually**
- **PARENT_OF**
- **PROPERTY**
- **REFERENCE**
- **WITH CHILDREN**
- **WITH CHILDREN ADD**

FMLINFO function
FOC144 message
- **eliminating with MULTIPATH parameter**

FOC2GIGDB
FOCERROR in Adabas Write
FOCFIRSTPAGE
FOCPROF - The System Wide Profile
FOCPSB
FOCSET
FOCSORT
- **allocating multiple units**
- **allocating multiple volumes**

FOCSTYLE files

FOCUS 7.0.9

NF575 Fusion
NF584 Dynamically Setting the IDMS DBNAME and DICTNAME
NF597 Aggregation by External Sort
NF652 Teradata Interface Kanji Support
NF654 HOLD From External Sort
NF655 FOCPROF - The System Wide Profile
NF656 Controlling REBUILD Messages
NF660 Multi-volume Support in MVS FOCUS
NF670 DYNAM Support for Unit Count
NF673 Model 204 Interface Account Split
NF691 Escape Character for LIKE
NF716 Euro Currency Support
NF718 DYNAM Support for Existing Relative GDG Numbers
NF720 SQLJOIN OUTER Setting for Relational Interfaces
NF722 FOCUS Client DNS Names Support
NF728 Changing Retrieval Order with Aggregation
NF730 Hold Format PDF
NF735 Enhancement to ? SET
NF740 Changes to the REBUILD PROMPT
NF744 HOLD FORMAT EXCEL
NF745 ? PTF Enhancements
NF746 Leading Zeros
NF748 HOLD FORMAT WP With Carriage Control

FOCUS 7.1

NF692 Aggregating and Sorting Report Columns
NF696 Calling Subroutines Written in REXX
NF731 Reporting From Independent Paths
NF749 HOLD FORMAT INTERNAL
NF750 DATASET in a Master File
NF751 Date-Time Data Type
NF755 Using FILEDEF for Creating Extract Files
NF759 Increased Number of Display Fields
NF761 Comma Suppress Edit Format Option
NF762 Percent Edit Format Option
NF766 DEFINE Functions
NF773 Token Delimited Files
NF777 Partitioned FOCUS data sources
NF777 Two-Gigabyte FOCUS Database Support
NF778 Dialogue Manager TRUNCATE Function
NF781 Embedding Text Fields in Headings
NF782 Oracle Data Adapter IXSPACE Setting
NF785 Adabas Write Data Adapter for FOCUS

FOCUS 7.2

NF551 AUTOIMS
NF646 WHERE-Based JOIN
NF767 Long Alphanumeric Fields
NF787 Increased Report Width
NF788 Tiles
NF789 FORECAST
NF790 ACROSS-TOTAL
NF792 SET CDN Enhancement
NF793 Displaying Fractions
NF796 Unlimited Nested -INCLUDE Commands
NF797 Field-Based Reformatting
NF801 DB2 Data Adapter SET SSID Command for MSO
NF802 Long Master File Names
NF817 Creating Comma-Delimited Files
NF824 STRIP
NF825 SET PCOMMA
NF826 SQUEEZ
NF827 TRIM
NF828 SUFFIX=COM/COMT/TABT
NF829 AUTODATE
NF833 Creating Tab-Delimited Files
NF849 Extended Currency Symbol Support
NF880 Exit on Error
NF886 Reusable MSO/CICS Control Blocks
NF898 SQL Support for Date-Time Formats
NF914 LMF Site Code Warning Message

FOCUS 7.3

NF771 Long Dialogue Manager Variables
NF830 Minus Edit Format Option
NF861 Increased Row Size Limit for Relational Data Adapters
NF866 DB2 FETCH FIRST n ROWS Support
NF888 Excel 2000
NF916 XFOCUS Data Source
NF917 Long Segment Names
NF920 NORMSDST and NORMSINV Subroutines
NF922 AnV Format
NF923 COMBINE Command Enhancement
NF926 Controlling Report Page Numbering
NF928 Compiled DEFINEs
NF929 Multivariate Linear Regression
NF930 Holding Missing Values
NF931 Multiple FOLD-LINE
NF932 FORMULTIPLE
NF935 Long Index Names
NF936 Summary Prefix Operators
NF937 Optimizing Selection of Relational Variable Length Character Data Types
NF938 Decimal Alignment of Headings
NF939 SET SAVEDMASTERS
NF940 SET HNODATA
NF943 Increased Number of ACROSS Columns
NF944 Increased Space for Column Titles
NF945 Compound Reports
NF946 &FOCUSER
NF951 FML Hierarchies

FOCUS 7.3 (continued)

NF952 TABLASTPAGE
NF953 File Extensions for Maintain Procedures
NF954 Enhanced Screening Conditions for Maintain
NF955 Long Qualified Field Names
NF956 Wide Lines
NF957 FIELDTYPE=R
NF958 StyleSheet Enhancements
NF959 DATASET for a Segment in a Master File
NF961 USERFCHK Setting
NF964 Extensions to FORECAST
NF965 Raised Limit for External Index
NF968 FMLINFO Function
NF970 SUFFIX=TAB
NF971 SET NULL=ON
NF972 NEWPAGE
NF973 Cascading Style Sheets
NF977 IN-RANGES-OF
NF979 SET HOLDFORMAT
NF980 SET BLANKINDENT
NF981 Increased Number of Sort Headings and Footings
NF982 SET BYDISPLAY
NF983 Indenting FML Reports
NF984 IEDIT Command
NF985 DYNAM Support for New GDG Iterations
NF986 GROUP Fields in FOCUS Data Sources
NF987 MDI for FOCUS and XFOCUS Data Sources
NF989 Excel 97
NF994 FOCUS SETs From Maintain
NF995 Raised Limit for IF-THEN-ELSE

FOCUS Client
  NF722 DNS Names Support
FOCUS database
  Access File
  allocating multiple units
  allocating multiple volumes
  GROUP fields
  MDI
  partitioned
  two-gigabyte
FOCUS referential integrity
  DELETE
  INCLUDE
FOCUS SETs from Maintain
FOCUS StyleSheets
  comments in
FOLD-LINE command
  multiple
FONT attribute
font map file entries
font map files
  editing in MVS
  PDF

fonts
- **colors**
- **configuring PostScript fonts**
- **default browser type**
- **default type**
- **in HTML Reports**
- **inherited styles**
- **measuring**
- **PostScript Type 1**
- **relative point sizes and HTML fonts**
- **sizes**

FOOTING component

footings
- **aligning**
- **embedded fields in**
- **identifying in a style sheet**
- **positioning**

FORECAST
- **applying conditional formatting**
- **applying conditional styling**
- **DOUBLEXP**
- **EXPAVE**
- **extensions to**
- **graphing**
- **methods**
- **MOVAVE**
- **REGRESS**
- **reporting techniques**
- **SEASONAL**
- **syntax**

format
  AnV
  date-time
  date-time ACTUAL
  date-time for relational data sources
  options for currency symbols
  overrides
formatted string date format
formatting heading and footing lines
formatting reports
  choosing style sheets
  methods
FORMULTIPLE
fractions
free text
  identifying in FML reports
free-form reports
freeing long Master File names
FREETEXT component

Functional Area (Data Adapter Enhancements)
- NF551 AUTOIMS
- NF584 Dynamically Setting the IDMS DBNAME and DICTNAME
- NF652 Kanji support
- NF673 Model 204 Interface Account Split
- NF720 SQLJOIN OUTER Setting
- NF782 Oracle Data Adapter IXSPACE Setting
- NF785 Adabas Write Data Adapter for FOCUS
- NF801 SET SSID Command for MSO
- NF861 Increased Row Size Limit for Relational Data Adapters
- NF866 DB2 FETCH FIRST n ROWS Support
- NF937 Optimizing Selection of Relational Variable Length Character Data Types
- NF957 FIELDTYPE=R

Functional Area (Database Enhancements)
- NF916 XFOCUS Data Source
- NF917 Long Segment Names
- NF935 Long Index Names
- NF953 File Extensions for Maintain Procedures
- NF954 Screening Conditions for Maintain
- NF959 DATASET for a Segment
- NF986 GROUP Fields in FOCUS Data Sources
- NF987 MDI for FOCUS and XFOCUS Data Sources
- NF994 FOCUS SETs From Maintain

Functional Area (FOCUS Client)
- NF722 FOCUS Client DNS Names Support

Functional Area (Fusion)
- NF575 Fusion

Functional Area (General Enhancements)

NF655 FOCPROF - The System Wide Profile
NF656 Controlling REBUILD Messages
NF660 Multi-volume Support in MVS FOCUS
NF670 DYNAM Support for Unit Count
NF696 Calling Subroutines Written in REXX
NF716 Euro Currency Support
NF718 DYNAM Support for Existing Relative GDG Numbers
NF735 Enhancement to ? SET
NF740 Changes to the REBUILD Prompt
NF745 ? PTF Enhancements
NF746 Leading Zeros
NF750 DATASET in a Master File
NF751 Date-Time Data Type
NF771 Long Dialogue Manager Variables
NF773 Token Delimited Files
NF777 Two-Gigabyte FOCUS Database Support
NF778 Dialogue Manager TRUNCATE Function
NF796 Unlimited Nested -INCLUDE Commands
NF802 Long Master File Names
NF824 STRIP Subroutine
NF825 SET PCOMMA
NF826 SQUEEZ
NF827 TRIM
NF828 SUFFIX=COM/COMT/TABT
NF829 AUTODATE
NF880 Exit on Error
NF898 SQL Support for Date-Time Formats

Functional Area (General Enhancements - continued)
- NF914 LMF Site Code Warning Message
- NF920 NORMSDST and NORMSINV Subroutines
- NF922 AnV Format
- NF939 SET SAVEDMASTERS
- NF946 &FOCUSER
- NF956 Wide Lines
- NF961 USERFCHK Setting
- NF970 SUFFIX=TAB
- NF984 IEDIT Command
- NF985 DYNAM Support for New GDG Iterations

Functional Area (MSO Enhancements)
- NF801 DB2 Data Adapter SET SSID Command
- NF886 Reusable MSO/CICS Control Blocks

Functional Area (Performance Enhancements)
- NF597 Aggregation by External Sort
- NF654 HOLD From External Sort
- NF728 Changing Retrieval Order with Aggregation
- NF777 Partitioned FOCUS Data Sources
- NF916 XFOCUS Data Source
- NF987 MDI for FOCUS and XFOCUS Data Sources

Functional Area (Raised Limits)
- NF759 Increased Number of Display Fields
- NF767 Long Alphanumeric Fields
- NF777 Two-Gigabyte FOCUS Database Support
- NF787 Increased Report Width
- NF923 COMBINE Command Enhancement
- NF943 Increased Number of ACROSS Columns
- NF944 Increased Space for Column Titles
- NF955 Increased Space for Column Titles
- NF965 Raised Limit for External Index
- NF981 Increased Number of Sort Headings and Footings
- NF995 Raised Limit for IF-THEN-ELSE Conditions

Functional Area (Reporting Enhancements)
- NF646 WHERE-Based JOIN
- NF691 Escape Character for LIKE
- NF692 Aggregating and Sorting Report Columns
- NF730 Hold Format PDF
- NF731 Reporting From Independent Paths
- NF744 HOLD FORMAT EXCEL
- NF748 HOLD FORMAT WP with Carriage Control
- NF749 HOLD FORMAT INTERNAL
- NF755 Using FILEDEF for Creating Extract Files
- NF759 Increased Number of Display Fields
- NF761 Comma Suppress Edit Format Option
- NF762 Percent Edit Format Option
- NF766 DEFINE Functions
- NF781 Embedding Text Fields in Headings
- NF788 Tiles

Functional Area (Reporting Enhancements - continued)

NF789 FORECAST
NF790 ACROSS-TOTAL
NF792 SET CDN Enhancement
NF793 Displaying Fractions
NF797 Field-Based Reformatting
NF817 Creating Comma-Delimited Files
NF830 Minus Edit Format Option
NF833 Creating Tab-Delimited Files
NF849 Extended Currency Symbol Support
NF888 Excel 2000
NF926 Controlling Report Page Numbering
NF928 Compiled DEFINEs
NF929 Multivariate Linear Regression
NF930 Holding Missing Values
NF931 Multiple FOLD-LINE
NF932 FORMULTIPLE
NF936 Summary Prefix Operators
NF938 Decimal Alignment of Headings
NF940 SET HNODATA
NF945 Compound Reports
NF952 TABLASTPAGE
NF958 StyleSheet Enhancements
NF964 Extensions to FORECAST
NF968 FMLINFO Function
NF971 SET NULL=ON
NF972 NEWPAGE
NF973 Cascading Style Sheets

Functional Area (Reporting Enhancements - continued)
NF977 IN-RANGES-OF
NF979 SET HOLDFORMAT
NF980 SET BLANKINDENT
NF982 SET BYDISPLAY
NF983 Indenting FML Reports
NF989 Excel 97
functions
argument verification
clearing
date-time
date-time component names
date-time for SQL Translator
DEFINE
FMLINFO
FOCUS
HADD
HCNVRT
HDATE
HDIFF
HDTTM
HGETC
HINPUT
HMIDNT
HNAME
HPART
HSETPT
HTIME
NORMSDST and NORMSINV
querying

Fusion
FUSREXX macro
CMS FILETYPE
compiling in CMS
date return value
integer return value
loading in CMS
multiple tokens in parameters
MVS DDNAME
parameters
search order

G

GDG
DYNAM support for existing relative numbers
new iterations of
General Enhancements (FOCUS 7.0.9)
NF655 FOCPROF - The System Wide Profile
NF656 Controlling REBUILD Messages
NF660 Multi-volume Support in MVS FOCUS
NF670 DYNAM Support for Unit Count
NF716 Euro Currency Support
NF718 DYNAM Support for Existing Relative GDG Numbers
NF735 Enhancement to ? SET
NF740 Changes to the REBUILD Prompt
NF745 ? PTF Enhancements
NF746 Leading Zeros

General Enhancements (FOCUS 7.1)
- NF696 Calling Subroutines Written in REXX
- NF750 DATASET in a Master File
- NF751 Date-Time Data Type
- NF773 Token Delimited Files
- NF777 Two-Gigabyte FOCUS Database Support
- NF778 Dialogue Manager TRUNCATE Function

General Enhancements (FOCUS 7.2)
- NF796 Unlimited Nested -INCLUDE Commands
- NF802 Long Master File Names
- NF824 STRIP Subroutine
- NF825 SET PCOMMA
- NF826 SQUEEZ Subroutine
- NF828 SUFFIX=COM/COMT/TABT
- NF829 AUTODATE
- NF880 Exit on Error
- NF886 Reusable MSO/CICS Control Blocks
- NF914 LMF Site Code Warning Message

General Enhancements (FOCUS 7.3)
- NF771 Long Dialogue Manager Variables
- NF920 NORMSDST and NORMSINV Subroutines
- NF922 AnV Format
- NF939 SET SAVEDMASTERS
- NF946 &FOCUSER
- NF956 Wide Lines
- NF961 USERFCHK Setting
- NF970 SUFFIX=TAB
- NF984 IEDIT Command
- NF985 DYNAM Support for New GDG Iterations

generating internal Cascading Style Sheets
GET CHILDREN
GET CHILDREN ADD
GRANDTOTAL component
graphic elements
  adding to a report
  linking
graphics, adding to a report
GRID attribute
GRID parameter
grids
GROUP attribute
GROUP fields in FOCUS data sources
GUTTER attribute

H

H format code
H USAGE format
HADD date-time function
HCNVRT date-time function
HDATE date-time function
HDIFF date-time function
HDTTM date-time function
HEADALIGN attribute
HEADING component

headings
  aligning
  embedding text in
  identifying in a style sheet
  positioning
HGETC date-time function
HGRID attribute
HINPUT date-time function
HMIDNT date-time function
HNAME date-time function
HNODATA
HOLD
  allocating with FILEDEF
  and date-time data type
  and long Master File names
  FORMAT EXCEL
  FORMAT INTERNAL
  Format PDF
  FORMAT WP with carriage control
  from external sort
HOLD FORMAT TABT
HOLDFORMAT
holding missing values
HOLDMISS
HPART date-time function
HSETPT date-time function
HTIME date-time function

HTML
  alignment
HTML format
HTML reports
  fonts in
  hyperlinks in
HTMLCSS SET parameter
hyperlinks

I

IBITABLA default space allocation
identifying report components
  columns
  data
  embedded fields
  free text
  headings and footings
  page numbers
  rows
  skipped lines
  sort values
  text strings
  titles
  totals and subtotals
  underlines
IEDIT Command
IF
  and date-time values

IMAGE attribute
images
  adding to a report
  linking
  StyleSheets and
  supplying descriptions for screen readers
IMS Data Adapter
  NF551 AUTOIMS
-INCLUDE
INCLUDE
increased
  number of ACROSS columns
  number of display fields
  number of sort headings and footings
  report width
  row size limit for relational data adapters
  space for column titles
INDENT
indenting FML Reports
independent paths
  and FOC144 message
  and SET ALL
  required segments
  resource requirements for MULTIPATH
index
  encoding
  names, long
index in MODIFY

index space parameters for Oracle
inheritance in style sheets
inheriting attributes
inline Style Sheets
IN-RANGES-OF
INSERT rules for Adabas data sources
inserting blank lines
inserting underlines
integer return value from REXX subroutine
intelligent partitioning
internal Cascading Style Sheets
ISO standards for date-time
ITEM attribute
IXSPACE

J

Japanese yen symbol
JavaScript functions
JOIN
  expression
  LOOKUP
  versus COMBINE

joins
  and Access Files
  and partitioned FOCUS data sources
  clearing
  conditional
  left outer
  listing
  to all dimensions in MDI
  to dimensions in MDI
  to MDI
  WHERE-based
justification regions
JUSTIFY attribute
  measuring for decimal alignment
justifying
  column titles
  data
  grand totals
  headings and footings
  subtotals

K

Kanji character set
KEEPDEFINES

L

LABEL attribute

LABELPROMPT attribute
labels
landscape orientation
last page number
LEADZERO
left margins
LEFTGAP attribute
LEFTMARGIN attribute
LIKE with escape character
limits
  alphanumeric field length
  data sources per MAINTAIN request
  data sources per MODIFY request
  display objects
  external index partitions
  FOCUS database size
  IF-THEN-ELSE conditions
  increased space for column title
  Master File name length
  nested -INCLUDE commands
  number of ACROSS columns
  number of display fields
  number of sort headings and footings
  qualified field names
  report width
LINE attribute
linear regression
  multivariate

line-by-line formatting
LINES SET parameter
linking report components
linking report pages
  heading text and
  images and
  in HTML reports
  page numbers and
linking reports
  setting conditions for
linking to external Cascading Style Sheets
linking to JavaScript functions
linking with conditions
LMF Site Code Warning Message
LOAD CHART
LOCATION attribute
  in Access File
logical sort page
logical unit of work
long
  alphanumeric fields
  Dialogue Manager Variables
  index names
  Master File names
  Master File names and HOLD
  Master File names and relational data sources
  qualified field names
  segment names

LONGNAME
LOOKUP

M

MACRO attribute
macros
  applying
  defining
  overriding
mailing labels
Maintain
  Adabas data sources
  CONTAINS
  DELETE
  enhanced screening conditions
  EQ_MASK
  file extensions
  INCLUDE
  limits
  MATCH
  NE_MASK
  NEXT
  non-updatable Adabas fields
  NOT_IN
  OMITS
  prerequisites
  processing overview
  referential integrity

Maintain (continued)
  SET commands
  SYS_MGR.FOCSET
  transaction control for Adabas Write
  UPDATE
  without unique keys
margins
Master File
  ACCESSFILE attribute
  DATASET attribute
  for Adabas Write
  generated by AUTOIMS
  long names
  sample for Adabas Write
Master Files, saving
MASTERNAME
MASTERNAME attribute
MATCH
  DELETE
  INCLUDE
  MAINTAIN
  UPDATE
MATRIXORDER attribute
MAXVALUES

MDI
- **AUTOINDEX**
- **building**
- **choosing dimensions**
- **creating**
- **DATANAME**
- **DCB attributes**
- **DIMENSION**
- **dimensional join**
- **encoding indexed values**
- **guidelines**
- **JOIN syntax**
- **joining to**
- **maintaining**
- **MASTERNAME**
- **MAXVALUES**
- **MDILOCATION**
- **partitioning**
- **query command**
- **querying progress**
- **using in a request**
- **warning message**
- **WITHIN**

MDICARDWARN
MDIENCODING
MDIFEATURE
MDILOCATION
measurement

measurement units
measuring
  for column width alignment
  for decimal alignment
messages for Adabas data source
metrics file, editing in MVS
minus edit format option
MISSING
  and date-time data type
missing values
  holding
  in comma and tab delimited files
  in HOLD file
Model 204 Data Adapter
  NF673 Model 204 Interface Account Split
MODIFY
  Adabas data sources
  case logic
  COMBINE
  COMBINE SUFFIX
  COMBINE versus JOIN
  COMMIT WORK
  DELETE
  differences from standard FOCUS
  FIND
  INCLUDE
  limits
  LOOKUP

MODIFY (continued)
- **MATCH**
- **NEXT**
- **non-updatable Adabas fields**
- **prerequisites**
- **processing overview**
- **referential integrity**
- **ROLLBACK WORK**
- **transaction control for Adabas Write**
- **UPDATE**
- **VALIDATE**
- **without unique keys**

MOVAVE
moving average
MSO
- **allocating multiple units**
- **allocating multiple volumes**
- **DB2 Data Adapter SET SSID Command**
- **reusable MSO/CICS control blocks**

MSO/CICS control blocks
multidimensional index
multi-pane reports
- **printing**

MULTIPATH parameter
- **and FOC144 message**
- **and SET ALL**
- **required segments**
- **resource requirements**

multiple
  FOLD-LINE commands
  units in MVS FOCUS
  volumes
  volumes and default allocations
multi-table HTML reports
multivariate linear regression
MVS batch
  allocating multi-volume data sources

N

names
  long
  qualified
naming CSS classes
NE_MASK
nested expressions, raised limit
New Features (FOCUS 7.0.9)
  ? PTF Enhancements
  Aggregation by External Sort
  Changes to the REBUILD PROMPT
  Changing Retrieval Order with Aggregation
  Controlling REBUILD Messages
  DYNAM Support for Existing Relative GDG Numbers
  DYNAM Support for Unit Count
  Dynamically Setting the IDMS DBNAME and DICTNAME
  Euro Currency Support
  FOCPROF - The System Wide Profile

New Features (FOCUS 7.0.9 - continued)
FOCUS Client DNS Names Support
Fusion
HOLD FORMAT EXCEL
Hold Format PDF
HOLD FORMAT WP With Carriage Control
HOLD From External Sort
Leading Zeros
Model 204 Interface Account Split
Multi-volume Support in MVS FOCUS
NF691 Escape Character for LIKE
NF735 Enhancement to ? SET
SQLJOIN OUTER Setting for Relational Interfaces
Teradata Interface Kanji Support
New Features (FOCUS 7.1)
Adabas Write Data Adapter for FOCUS
Aggregating and Sorting Report Columns
Calling Subroutines Written in REXX
Comma Suppress Edit Format Option
DATASET in a Master File
Date-Time Data Type
DEFINE Functions
Dialogue Manager TRUNCATE Function
Embedding Text Fields in Headings
HOLD FORMAT INTERNAL
Increased Number of Display Fields
Oracle Data Adapter IXSPACE Setting
Partitioned FOCUS Data Sources

New Features (FOCUS 7.1 - continued)
- Percent Edit Format Option
- Reporting From Independent Paths
- Token Delimited Files
- Two-Gigabyte FOCUS Database Support
- Using FILEDEF for Creating Extract Files

New Features (FOCUS 7.2)
- ACROSS-TOTAL
- AUTODATE
- AUTOIMS
- Creating Comma-Delimited Files
- Creating Tab-Delimited Files
- DB2 Data Adapter SET SSID Command for MSO
- Displaying Fractions
- Exit on Error
- Extended Currency Symbol Support
- Field-Based Reformatting
- FORECAST
- Increased Report Width
- LMF Site Code Warning Message
- Long Alphanumeric Fields
- Long Master File Names
- Unlimited Nested -INCLUDE Commands
- Reusable MSO/CICS Control Blocks
- SET CDN Enhancement
- SET PCOMMA
- SQL Support for Date-Time Formats
- SQUEEZ

New Features (FOCUS 7.2 - continued)
- STRIP Subroutine
- SUFFIX=COM/COMT/TABT
- Tiles
- TRIM Subroutine
- Unlimited Nested -INCLUDE Commands
- WHERE-Based JOIN

New Features (FOCUS 7.3)
- &FOCUSER
- AnV Format
- Cascading Style Sheets
- COMBINE Command Enhancement
- Compiled DEFINEs
- Compound Reports
- Controlling Report Page Numbering
- DATASET for a Segment
- DB2 FETCH FIRST n ROWS Support
- Decimal Alignment of Headings
- DYNAM Support for New GDG Iterations
- Enhanced Screening Conditions for Maintain
- Excel 2000
- Excel 97
- Extensions to FORECAST
- FIELDTYPE=R
- File Extensions for Maintain Procedures
- FML Hierarchies
- FMLINFO Function
- FOCUS SETs From Maintain

New Features (FOCUS 7.3 - continued)

FORMULTIPLE
GROUP Fields in FOCUS Data Sources
Holding Missing Values
IEDIT Command
Increased Number of ACROSS Columns
Increased Number of Sort Headings and Footings
Increased Row Size Limit for Relational Data Adapters
Increased Space for Column Titles
Indenting FML Reports
IN-RANGES-OF
Long Dialogue Manager Variables
Long Index Names
Long Qualified Field Names
Long Segment Names
MDI for FOCUS and XFOCUS Data Sources
Minus Edit Format Option
Multiple FOLD-LINE
Multivariate Linear Regression
NEWPAGE
NORMSDST and NORMSINV Subroutines
Optimizing Selection of Relational Variable Length Character Data Types
Raised Limit for External Index
Raised Limit for IF-THEN-ELSE Conditions
SET BLANKINDENT
SET BYDISPLAY
SET HNODATA
SET HOLDFORMAT

New Features (FOCUS 7.3 - continued)
SET NULL=ON
SET SAVEDMASTERS
StyleSheet Enhancements
SUFFIX=TAB
Summary Prefix Operators
TABLASTPAGE
USERFCHK Setting
Wide Lines
XFOCUS Data Source
NEWPAGE
NEXT
MAINTAIN
non-unique key in Adabas Write
unique key in Adabas Write
without MATCH
NOCC
normal distribution
NORMSDST and NORMSINV subroutines
NOT_IN
NULL
number of last page
numbers for pages
numeric string date format

O

OBJECT attribute
OMITS

optimization
  and date-time values
  of outer joins
optimizing selection of relational variable length character data types
options for Adabas Write
Oracle Data Adapter IXSPACE Setting
Oracle data sources
  and date-time data type
  and long Master File names
ORIENTATION attribute
outer join
  optimization
OVER command

P

padding, preventing in HOLD files
page color
page colors
page count
page margins
page numbers
  for last page
page orientation
page size
page, size for XFOCUS data sources
PAGECOLOR attribute
PAGEMATRIX attribute
PAGENUM component

PAGESIZE attribute
parameters, verifying in functions
PARENT_OF
partitioned FOCUS data sources
partitioning an MDI
partitions
  and joins
  FOCUS database
  intelligent
paths, independent
  and FOC144 message
  and SET ALL
  required segments
  resource requirements for MULTIPATH
PCOMMA
PDF
  compound reports
  font map file entries
  format
PDF format
percent edit format option
Performance Enhancements (FOCUS 7.0.9)
  NF597 Aggregating by External Sort
  NF654 HOLD From External Sort
  NF728 Changing Retrieval Order with Aggregation
Performance Enhancements (FOCUS 7.1)
  NF777 Partitioned FOCUS Data Sources

Performance Enhancements (FOCUS 7.3)
  NF916 XFOCUS Data Source
  NF987 MDI for FOCUS and XFOCUS Data Sources
personal Cascading Style Sheets
portrait orientation
POSITION attribute
positioning columns
positioning headings and footings
positioning report components
PostScript (PS) files
  compound reports
PostScript fonts
PostScript format
pound symbol
prefix operators
  with SUBTOTAL
printing
  labels
  multi-pane reports
  StyleSheets
PROPERTY
PS (PostScript) files
  compound reports
PS format
PTF query command

Q

qualified field names

query commands
? FUNCTION
? MDI
? MDIPROGRESS
? PTF
? STYLE
QUOTE

R

Raised Limits (FOCUS 7.1)
NF759 Increased Number of Display Fields
NF777 Two-Gigabyte FOCUS Database Support
Raised Limits (FOCUS 7.2)
NF767 Long Alphanumeric Fields
NF787 Increased Report Width
Raised Limits (FOCUS 7.3)
NF923 COMBINE Command Enhancement
NF943 Increased Number of ACROSS Columns
NF944 Increased Space for Column Titles
NF955 Long Qualified Field Names
NF965 Raised Limit for External Index
NF981 Increased Number of Sort Headings and Footings
NF995 Raised Limit for IF-THEN-ELSE Conditions
ranges
RDBMS tables
and long Master File names
REBUILDMSG
RECAP component

RECFM VBA
REFERENCE
referential integrity
  and COMBINE
  FOCUS DELETE
  FOCUS INCLUDE
  inhibiting
refreshing external Cascading Style Sheets
REGRESS
regression
Relational Data Adapters
Relational Data Adapters (FOCUS 7.0.9)
  NF652 Teradata Kanji Support
  NF720 SQLJOIN OUTER Setting
Relational Data Adapters (FOCUS 7.1)
  NF782 Oracle Data Adapter IXSPACE Setting
Relational Data Adapters (FOCUS 7.2)
  NF801 DB2 Data Adapter SET SSID Command for MSO
  NF898 Date-Time Format Support
Relational Data Adapters (FOCUS 7.3)
  NF861 Increased Row Size
  NF866 DB2 V7.1 FETCH FIRST n ROWS Support
  NF937 Selection of Variable Length Character Data Types
  NF957 FIELDTYPE=R
relational data sources
  and long Master File names
relative starting positions
REPORT

report
  increased display fields
  number of display fields in
report components
  linking
  positioning
report display formats
  Excel 2000
  Excel 97
  PDF
  PostScript
  PS
report formatting
  choosing style sheets
  compound reports
  external Cascading Style Sheets
  inheritance in style sheets
  internal Cascading Style Sheets
  methods
  tabular reports
report headings
  embedding text in
  text alignment
report styling
  external Cascading Style Sheets
report titles
report width, increased

Reporting Enhancements (FOCUS 7.0.9)
- NF691 Escape Character for LIKE
- NF730 Hold Format PDF
- NF744 HOLD FORMAT EXCEL
- NF748 HOLD FORMAT WP With Carriage Control

Reporting Enhancements (FOCUS 7.1)
- NF692 Aggregating and Sorting Report Columns
- NF731 Reporting From Independent Paths
- NF749 HOLD FORMAT INTERNAL
- NF755 Using FILEDEF for Creating Extract Files
- NF759 Increased Number of Display Fields
- NF761 Comma Suppress Edit Format Option
- NF762 Percent Edit Format Option
- NF766 DEFINE Functions
- NF781 Embedding Text Fields in Headings

Reporting Enhancements (FOCUS 7.2)
- NF646 WHERE-Based JOIN
- NF788 Tiles
- NF789 FORECAST
- NF790 ACROSS-TOTAL
- NF792 SET CDN Command Enhancement
- NF793 Displaying Fractions
- NF817 Creating Comma-Delimited Files
- NF827 TRIM Subroutine
- NF833 Creating Tab-Delimited Files
- NF849 Extended Currency Symbol Support

Reporting Enhancements (FOCUS 7.3)
- **NF830 Minus Edit Format Option**
- **NF888 Excel 2000**
- **NF926 Controlling Report Page Numbering**
- **NF928 Compiled DEFINEs**
- **NF929 Multivariate Linear Regression**
- **NF930 Holding Missing Values**
- **NF931 Multiple FOLD-LINE**
- **NF932 FORMULTIPLE**
- **NF936 Summary Prefix Operators**
- **NF938 Decimal Alignment of Headings**
- **NF940 SET HNODATA**
- **NF945 Compound Reports**
- **NF951 FML Hierarchies**
- **NF952 TABLASTPAGE**
- **NF958 StyleSheet Enhancements**
- **NF964 Extensions to FORECAST**
- **NF968 FMLINFO Function**
- **NF971 SET NULL=ON**
- **NF972 NEWPAGE**
- **NF973 Cascading Style Sheets**
- **NF977 IN-RANGES-OF**
- **NF979 SET HOLDFORMAT**
- **NF980 SET BLANKINDENT**
- **NF982 SET BYDISPLAY**
- **NF983 Indenting FML Reports**
- **NF989 Excel 97**

reporting from independent paths

reports
- **comparing styled and non-styled**
- **customizing**
- **displaying**
- **displaying in a browser**
- **formatting with StyleSheets**
- **justifying columns**
- **linking**
- **positioning headers and footers**
- **specifying fonts for**
- **styling**
- **wrapping data**
- **wrapping with alternative methods**

REPOSITION

requests
- **increased display fields**
- **number of display fields in**

return code on error

Reusable MSO/CICS Control Blocks

REXX user-written subroutines
- **CMS FILETYPE**
- **compiling in CMS**
- **date return value**
- **integer return value**
- **loading in CMS**
- **multiple tokens in parameters**
- **MVS DDNAME**
- **parameters**
- **search order**

right margins
RIGHTGAP attribute
RIGHTMARGIN attribute
ROLLBACK WORK
row titles
rows
  identifying in a style sheet
ROWTOTAL attribute
rules
  for delete from Adabas data sources
  for update of Adabas data sources
  in Cascading Style Sheets
  insert for Adabas data sources
rules in Cascading Style Sheets

S

SAVE files
  and date-time data type
SAVEDMASTERS
screening conditions
  for Maintain
  in a join
SEASONAL
segment
  DATASET attribute
  names, long
  SYSTEM99
segments, required for report

SEGTYPE for Adabas Write
SEQUENCE attribute
sequential conditional formatting
SET
- **ACCTNAME**
- **ACCTPASS**
- **ALL and MULTIPATH**
- **AUTOINDEX**
- **BLANKINDENT**
- **BYDISPLAY**
- **CDN**
- **CENT-ZERO**
- **DATEFORMAT**
- **DBNAME**
- **DICTNAME**
- **DTSTRICT**
- **ERROROUT**
- **ERRORRUN**
- **EUROFILE**
- **EXTAGGR**
- **EXTHOLD**
- **FOC2GIGDB**
- **from Maintain**
- **GRID**
- **HNODATA**
- **HOLDFORMAT**
- **HOLDMISS**
- **HTMLCSS**

SET (continued)
- **IXSPACE (Oracle)**
- **LEADZERO**
- **LINES**
- **MDICARDWARN**
- **MDIENCODING**
- **MDIFEATURE**
- **MULTIPATH**
- **NULL**
- **PCOMMA**
- **SAVEDMASTERS**
- **SPACES**
- **SQLJOIN OUTER**
- **SQUEEZE**
- **SSID**
- **STYLEMODE**
- **STYLESHEET (STYLE)**
- **SUMPREFIX**
- **TRANTERM**
- **UNITS**
- **USERFCHK**
- **USERFNS**
- **WEEKFIRST**
- **XFOCUS**
- **XFOCUSBINS**

SET CDN enhancement
SET CENT-ZERO
sheet names

simple moving average
site code warning message
SIZE attribute
SKIPLINE attribute
SKIPLINE component
SKIP-LINE option
skipped lines
  formatting
sort footings, increased limit
sort headings
  aligning
  increased limit
sort values, identifying
sorting
  by a calculated value
  controlling retrieval order
SPACE
SPACES SET parameter
spacing between columns
specifying URLs
spot markers
SQL
  COMMIT WORK
  ROLLBACK WORK
  SET IXSPACE (Oracle)
  support for date-time formats
SQL support for date-time formats

SQL Translator
  date-time functions
  support for date-time formats
SQL/DS data source and date-time data type
SQLJOIN OUTER
SQUEEZ
SQUEEZE attribute
SQUEEZE parameter
SSID
stacking columns
STRIP
STYLE
STYLE attribute
style sheet attributes
style sheets
  Cascading Style Sheets (CSS)
  conditional styling and
  declarations
  FOCUS
  FOCUS StyleSheets
  graphics and
  inheritance
  macros
  multi-pane reports and
  refreshing external Cascading Style Sheets
  selecting
  troubleshooting
styled reports

STYLEMODE SET parameter
StyleSheet attributes
  CLASS
  CSSURL
StyleSheet declarations
StyleSheet enhancements
StyleSheet files
StyleSheets
  activating
  adding graphics and
  CHECK STYLE
  conditional styling
  FOCSTYLE file
  identifying report components
  inheritance
  linking reports with
  positioning components
  printing
  REPORT
  report components
  requirements
  StyleSheet file
  WHEN
styling reports
  external Cascading Style Sheets
  free-form reports
SUBFOOT component
SUBFOOT, increased limit

SUBHEAD component
SUBHEAD, increased limit
subheadings, embedding text in
subroutines
  argument verification
  FMLINFO
  NORMSDST and NORMSINV
  REXX
  REXX, CMS FILETYPE
  REXX, compiling in CMS
  REXX, date return value
  REXX, integer return value
  REXX, loading in CMS
  REXX, multiple tokens in parameters
  REXX, MVS DDNAME
  REXX, parameters
  REXX, search order
  SQUEEZ
  STRIP
  TRIM
SUBTOTAL
  with alphanumeric field
  with prefix operators
SUBTOTAL component
subtotals
  identifying in a style sheet

SUFFIX
  COM/COMT/TABT
  COMBINE
  for XFOCUS data sources
  TAB
summary prefix operators
  displaying all fields
SUMPREFIX
synonyms in Adabas Write
SYS_MGR.FOCSET
system editor
SYSTEM99

T

tab delimited files
TAB suffix
tab-delimited files
TABFOOTING component
TABHEADING component
TABLA
TABLASTPAGE
TABLASTPAGE system variable
TABLE
  language formatting
  StyleSheets
TABT HOLD format
TABT suffix
tabular reports

target frames
TARGETFRAME SET parameter
TD element
Teradata data sources
  and long Master File names
  Kanji support
TEXT component
text fields
  aligning
  alignment in headings
  embedding in headings
  formatting
text strings
tiles
TIME format
timestamp
  for FOCUS data sources
TIMESTAMP format
TIMESTAMP support for DB2
TITLE component
titles
TITLETEXT attribute
token delimited files
top margins
TOPGAP attribute
TOPMARGIN attribute
total for ACROSS
total page count

totals
  identifying in a style sheet
transaction control in Adabas Write
  COMMIT WORK
  ROLLBACK WORK
translated string date format
TRANTERM parameter
TRIM subroutine
troubleshooting Cascading Style Sheets
TRUNCATE
TSO
  allocating multiple units
  allocating multiple volumes
Two-Gigabyte FOCUS Database Support
TYPE attribute in StyleSheet
  REPORT

U

U.S. dollar sign
UCOUNT
  allocating multiple units
UNDERLINE attribute
UNDERLINE component
UNDER-LINE option
underlines
  adding
  deleting
underlining values

Uniform Resource Locators (URLs)
  linking to
  specifying
unique key
  and INSERT for Adabas data sources
  in Adabas Write
unit of work
units
  allocating multiple
UNITS attribute
units of measurement
UNITS parameter
Unlimited Nested -INCLUDE Commands
UNQKEYNAME
  and INSERT for Adabas data source
UPDATE
  rules for Adabas data sources
URLs (Uniform Resource Locators)
  linking to
  specifying
USAGE, date-time (H)
user errors for Adabas Write
USERFCHK Setting
USERFNS
using FILEDEF for creating extract files

V

VALIDATE LOOKUP

VARGRAPHIC
variable length character data types, optimizing selection of
variables
  &FOCNEXTPAGE
  &FOCUSER
  long
  TABLASTPAGE
VBA
verb objects, 1024
vertical spacing
VGRID attribute
virtual fields
  preserving
virtual segment SYSTEM99
VM
  FILEDEF for creating extract files
VOLSER, multiple
volume, allocating multiple

W

warning message for MDI
Web browser support for Cascading Style Sheets
WEEKFIRST parameter
WHEN command
  ACROSSCOLUMN

WHERE test
  and date-time values
  in Access File
  in joins
WHERE-based JOIN
  AT attribute
  Master File syntax
  preserving virtual fields
wide Lines
  editing
WIDTH attribute
  measuring
width of columns
window titles
WITH CHILDREN
WITH CHILDREN ADD
WITHIN
worksheet titles
WP format
  with carriage control
WRAP attribute
WRAP parameter
wrapping data
  alternative methods
  by Web browser functionality
  using a SET command

write
- Access File for Adabas
- component for data source maintenance
- Master File for Adabas
- options for Adabas

X

XFOCUS data sources
- creating
- DCB attributes
- SUFFIX

XFOCUSBINS

Y

yen symbol