Information Builders

FOCUS for Mainframe

Summary of New Features

Version 7.6

DN1001076.0408

Contents

Preface

This manual describes the FOCUS new features for Version 7.6. This document is intended for all levels of users, including application developers, administrators, and end users.

How This Manual Is Organized

This manual includes the following chapters:

Chapter/Appendix		**Contents**
1	Performance Enhancements	❏ CPU Performance Improvement ❏ Unlimited FOCSORT ❏ Direct HOLD FORMAT FOCUS ❏ Compiling MODIFY Expressions Using Native Arithmetic
2	Reporting Enhancements	❏ Comparing Alphanumeric Fields to a Pattern With IF ❏ Assigning Column Numbers Only to Fields Displayed on Report Output ❏ Assigning Column Numbers to All Fields Used in a Request ❏ Controlling Underlining of ACROSS Objects ❏ Combinations of Summary Commands ❏ Producing Summary Columns for Horizontal Sort Fields ❏ Joining to a Fixed Format Sequential File Based on Multiple Fields ❏ Ranking Sort Field Values With the RNK. Prefix Operator ❏ Displaying a Row for Data Excluded by a Sort Phrase (PLUS OTHERS) ❏ SET EMPTYREPORT=ANSI

Chapter/Appendix		Contents
2	Reporting Enhancements (*continued*)	❑ Preserving Filter Definitions During Join Parsing ❑ SET SUMMARYLINES=EXPLICIT ❑ ON TABLE SET BYPANEL ❑ Left Outer Join Support ❑ FMLCAP Function ❑ FMLFOR Function ❑ FMLLIST Function ❑ Changing Titles of Columns Created Using Prefix Operators ❑ Joining Comma-Delimited Files ❑ Compiling DEFINEs With Missing Attributes ❑ SUBFOOT WITHIN

Chapter/Appendix		Contents
3	Output Format Enhancements	❏ Saving Report Output in PowerPoint Format ❏ Preserving Leading and Internal Blanks in Report Output ❏ Structured HOLD Files ❏ SAVB FORMAT INTERNAL ❏ Excel Named Ranges ❏ Identifying Null Values in EXL2K Format Reports ❏ Creating PDF Files for Use With UNIX Systems ❏ Excel Table of Contents ❏ Excel Compound Reports ❏ Displaying An and AnV Fields With Line Breaks ❏ Creating HTML Reports With Absolute Positioning ❏ Excel Cell Locking

Chapter/Appendix		Contents
4	General Enhancements	❏ DB_LOOKUP: Retrieving a Value From a Lookup Data Source ❏ HEXTR and HMASK Date-Time Functions ❏ Specifying Precision for Dialogue Manager Calculations ❏ Displaying SET Parameters by Functional Area ❏ New QUOTEP Option for Continental Decimal Notation ❏ Establishing a Default Value for the &ECHO Variable ❏ Retrieving the Site Code of the Connected User ❏ Establishing a Non-Overridable User Password ❏ Creating a Standard Quote-Delimited String ❏ Assignment Between Text Fields and Alphanumeric Fields ❏ DATETRAN Function ❏ Extensions to Date-Time Formats ❏ MIRR Function ❏ XIRR Function ❏ PTOA Function ❏ PUTDDREC and CLSDDREC Functions ❏ SET ERROROUT = OVERRIDE ❏ STRREP Function ❏ FOCREPLAY: Recording and Playing Back a FOCUS Session ❏ &FOCFEXNAME Variable ❏ Controlling Missing Values in Reformatted Fields ❏ Controlling Case Sensitivity of Passwords

Chapter/Appendix		Contents
4	General Enhancements (continued)	❏ SLEEP Function ❏ Setting a Currency Symbol for the M and N Format Options ❏ ISO Standard Week Numbering and the HYYWD Function ❏ PATTERN Function ❏ REVERSE Function ❏ XTPACK Function ❏ Alternate Extended Currency Symbol ❏ SET SUWEDGE
5	Database Enhancements	❏ Declaring Filters in a Master File ❏ Describing GROUP Fields as a Set of Elements ❏ Specifying Multilingual Descriptions in a Master File ❏ Specifying Multilingual TITLE Attributes in a Master File ❏ Using Date System Amper Variables in Master File DEFINEs ❏ CREATE FILE DROP ❏ COMPUTE in a Master File ❏ Comma-delimited Access File ❏ 1022 Partitions
6	MODIFY Enhancements	❏ Loading Fixed Format Sequential Files Using MODIFY ❏ MODIFY FIXFORM Support for Multiple Text Fields ❏ Controlling Whether FIXFORM Input Fields Are Conditional

Chapter/Appendix		Contents
7	Adapter Enhancements	❑ Adabas Dynamic CALLTYPE Setting ❑ Adabas FETCHJOIN Setting ❑ Adabas SQL NULL Option ❑ PASSRECS Setting for Adapters for Adabas, IMS, and VSAM ❑ VSAM RRDS Support ❑ HOLD FORMAT SAME_DB ❑ Adapter for Oracle: Increased Length of Column Descriptions ❑ Oracle 10g Support ❑ Relational Adapters: Controlling Optimization of Calculations ❑ Calling a Teradata Macro or Stored Procedure Using SQL Passthru
8	Raised Limits	❑ Increased Number of Segments in a Structure ❑ Increased Length for HOLD Files in FOCUS Format ❑ Increased Name Length for Dialogue Manager Variables ❑ Increased Number of Amper Variables ❑ Raised Limit for Number of FML Rows ❑ Increased Number of Joins ❑ Increased Number of Files in a MATCH FILE Request ❑ Increased Length for DESCRIPTION Attribute in a Master File ❑ Increase to the Total Length of All fields ❑ Increased Length of DBA Passwords ❑ SET MAXLRECL = 65536

Chapter/Appendix		Contents
9	Logging FOCUS Usage: FOCLOG	❑ FOCLOG: A facility for logging FOCUS activity using minimal resources.

Documentation Conventions

The following conventions apply throughout this manual:

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

Related Publications

To view a current listing of our publications and to place an order, visit our World Wide Web sites, http://www.informationbuilders.com or http://www.iwaysoftware.com. You can also contact the Publications Order Department at (800) 969-4636.

Customer Support

Do you have questions about FOCUS?

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

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

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

Information You Should Have

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

- Your six-digit site code (*xxxx.xx*).
- The stored procedure (preferably with line numbers) or FOCUS commands being used.
- The name of the Master File and Access File.
- The exact nature of the problem:
  - Are the results or the format incorrect? Are the text or calculations missing or misplaced?
  - The error message and return code, if applicable.
  - Is this related to any other problem?

- ❏ Has the procedure or query ever worked in its present form? Has it been changed recently? How often does the problem occur?
- ❏ What release of the operating system are you using? Has it, FOCUS, your security system, communications protocol, or front-end software changed?
- ❏ Is this problem reproducible? If so, how?
- ❏ Have you tried to reproduce your problem in the simplest form possible? For example, if you are having problems joining two data sources, have you tried executing a query containing the code to access a single data source?
- ❏ Do you have a trace file?
- ❏ How is the problem affecting your business? Is it halting development or production? Do you just have questions about functionality or documentation?

User Feedback

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

Thank you, in advance, for your comments.

Information Builders Consulting and Training

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

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

1 Performance Enhancements

This chapter describes features that enhance FOCUS performance.

Topics:

- ❑ CPU Performance Improvement
- ❑ Unlimited FOCSORT
- ❑ Direct HOLD FORMAT FOCUS
- ❑ Compiling MODIFY Expressions Using Native Arithmetic

CPU Performance Improvement

FOCUS Release 7.6 is the first release to use the IBM/C compiler to achieve performance improvements in CPU consumption for mainframe applications on z/OS and z/VM. CPU performance appears to benchmark from 5% to over 40% better in Release 7.6 than the prior release, 7.3, and appears to be roughly equivalent to statistics as far back as Release 7.0.8R released in 1999.

The benchmark tests were designed to approximate parts of a “typical” customer application to project real world expectations. From the test details, FOCUS customers can extrapolate the benchmark results to their own applications to approximate what CPU gains they may achieve.

Compared to Release 7.3.9, expect 5-10% improvement in CPU utilization for data extraction and reporting (TABLE), and 30-40% improvement for database modification procedures (MODIFY and MAINTAIN). Data intensive utilities such as REBUILD that typically absorb large amounts of CPU show about 40% improvement. Local environmental factors may affect the individual savings attained, and though different CPU models perform at varying speeds, the ratio of performance gains should be similar.

The wide range of performance improvements across the various FOCUS functional areas reflect the ratio of Assembler versus C code used by the FOCUS product for the selected function. For example, reporting (TABLE) employs a large percentage of fast-executing Assembler code during the expensive extraction phase, so the new C compiler has little effect on CPU utilization for reporting. Database updates (MODIFY and MAINTAIN), however, are almost entirely written in C so the savings are measurably larger.

For complete details of the benchmark tests and results, see the *Mainframe FOCUS Release 7.6 CPU Performance Improvement Benchmark* white paper on the Information Builders Web site at:

http://www.informationbuilders.com/products/whitepapers/pdf/Mainframe_FOCUS_Rel_7.6_WP.pdf

Unlimited FOCSORT

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

Direct HOLD FORMAT FOCUS

How to:

Control Whether a HOLD File in FOCUS Format is Created Directly

By default, a HOLD file in FOCUS format is now created directly, without using an intermediate sequential file and an internally generated MODIFY procedure to load the file, as in previous releases.

The SET DIRECTHOLD command controls whether the HOLD file is created directly or whether the HOLD file is loaded using an internally generated MODIFY procedure and an intermediate sequential file called FOC$HOLD.

Syntax: How to Control Whether a HOLD File in FOCUS Format is Created Directly

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

Use the following syntax to issue the SET DIRECTHOLD command in a request.

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

where:

`ON`

Creates a FOCUS HOLD file directly without an intermediate sequential file and MODIFY procedure. ON is the default value.

`OFF`

Loads a FOCUS HOLD file using an internally generated MODIFY procedure and an intermediate sequential file called FOC$HOLD.

Compiling MODIFY Expressions Using Native Arithmetic

How to:

Control Compilation of MODIFY Expressions

Reference:

Usage Notes for SET MODCOMPUTE

The native compiler for MODIFY processes COMPUTE, IF, and VALIDATE expressions using the arithmetic operations built into the underlying operating system. This native compiler eliminates internal format conversions and speeds up expression processing. It significantly enhances the speed of expressions that use long packed fields and date fields.

Note that MODCOMPUTE replaces the functionality previously provided by SET COMPUTE for MODIFY. SET COMPUTE is now only respected in the TABLE environment.

Note: Expression compilers for MODIFY are supported only in Mainframe environments. Linux on the Mainframe does not support these compilers.

Syntax: How to Control Compilation of MODIFY Expressions

```
SET MODCOMPUTE={NATV|NEW|OLD}
```

where:

`NATV`

Activates the native compiler for MODIFY expressions. NATV is the default value.

`NEW`

Compiles MODIFY expressions using the standard FOCUS compilation routines, which use high-precision floating point format for all arithmetic operations.

`OLD`

Does not compile MODIFY expressions.

Reference: Usage Notes for SET MODCOMPUTE

The following are usage notes for SET MODCOMPUTE:

- SET MODCOMPUTE can be issued in a user or system profile or on the command line.
- SET MODCOMPUTE is supported with compiled and uncompiled MODIFY procedures. Expression compilation is different from and compatible with MODIFY procedure compilation.
- Existing compiled MODIFY procedures run without recompilation. The MODCOMPUTE setting has no effect on previously compiled MODIFY procedures. In order to make use of this performance enhancement, compiled MODIFYs must be recompiled with SET MODCOMPUTE=NATV in effect.
- Expressions using the following features are not compiled by the native compiler:

  LIKE operator.

  DEFINE functions.

  LAST function.

2 Reporting Enhancements

This chapter describes features that affect reporting applications. For additional information, see Chapter 8, *Raised Limits* and Chapter 3, *Output Format Enhancements*

Topics:

- ❏ Comparing Alphanumeric Fields to a Pattern With IF
- ❏ Assigning Column Numbers Only to Fields Displayed on Report Output
- ❏ Assigning Column Numbers to All Fields Used in a Request
- ❏ Controlling Underlining of ACROSS Objects
- ❏ Combinations of Summary Commands
- ❏ Producing Summary Columns for Horizontal Sort Fields
- ❏ Joining to a Fixed Format Sequential File Based on Multiple Fields
- ❏ Ranking Sort Field Values With the RNK. Prefix Operator
- ❏ Displaying a Row for Data Excluded by a Sort Phrase (PLUS OTHERS)
- ❏ SET EMPTYREPORT=ANSI
- ❏ Preserving Filter Definitions During Join Parsing
- ❏ SET SUMMARYLINES=EXPLICIT
- ❏ ON TABLE SET BYPANEL
- ❏ Left Outer Join Support
- ❏ FMLCAP Function
- ❏ FMLFOR Function
- ❏ FMLLIST Function
- ❏ Changing Titles of Columns Created Using Prefix Operators
- ❏ Joining Comma-Delimited Files
- ❏ Compiling DEFINEs With Missing Attributes
- ❏ SUBFOOT WITHIN

Comparing Alphanumeric Fields to a Pattern With IF

In this section:

Using Escape Characters for LIKE and UNLIKE

How to:

Screen Using LIKE and UNLIKE in an IF Phrase

Example:

Screening on Initial Characters

Screening on a Single Character

The IF phrase can specify the LIKE and UNLIKE operators to compare alphanumeric fields to a pattern. The pattern used for the comparison is called a *mask*.

The mask can use the following wildcard characters:

- The percent sign (%), which accepts any following sequence of zero or more characters as a match to the pattern.
- The underscore character (_), which accepts any single character in that position as a match to the pattern.

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

To search for records with the LIKE operator, use

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

To reject records based on the mask value, use

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

where:

field

Is any valid field name or alias.

mask1, mask2

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

Example: Screening on Initial Characters

To list all employees who have taken basic-level courses, where every basic course begins with the word BASIC, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT COURSE_NAME COURSE_CODE
BY LAST_NAME BY FIRST_NAME
IF COURSE_NAME LIKE 'BASIC%'
END
```

The output is:

```
LAST_NAME        FIRST_NAME  COURSE_NAME                    COURSE_CODE
---------        ----------  -----------                    -----------
BLACKWOOD        ROSEMARIE   BASIC REPORT PREP NON-PROG     102
CROSS            BARBARA     BASIC REPORT PREP DP MGRS      107
JONES            DIANE       BASIC REPORT PREP FOR PROG     103
SMITH            MARY        BASIC REPORT PREP FOR PROG     103
                 RICHARD     BASIC RPT NON-DP MGRS          108
```

To list all employees who have taken higher level courses which do not begin with the word BASIC, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT COURSE_NAME COURSE_CODE
BY LAST_NAME BY FIRST_NAME
IF COURSE_NAME UNLIKE 'BASIC%'
END
```

The output is:

```
LAST_NAME        FIRST_NAME  COURSE_NAME                    COURSE_CODE
---------        ----------  -----------                    -----------
BLACKWOOD        ROSEMARIE   DECISION SUPPORT WORKSHOP      301
                             WHAT'S NEW IN FOCUS            202
                             TIMESHARING WORKSHOP           106
                             FILE DESC & MAINT NON-PROG     104
CROSS            BARBARA     HOST LANGUAGE INTERFACE        302
JONES            DIANE       FOCUS INTERNALS                203
                             ADVANCED TECHNIQUES            201
                             FILE DESCRPT & MAINT           101
MCKNIGHT         ROGER       FILE DESCRPT & MAINT           101
SMITH            MARY        FILE DESCRPT & MAINT           101
STEVENS          ALFRED      FILE DESCRPT & MAINT           101
```

Example: Screening on a Single Character

If you want to list all employees who have taken a 20x-series course, and you know that all of these courses have the same code except for the final character, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT COURSE_NAME COURSE_CODE
BY LAST_NAME BY FIRST_NAME
  IF COURSE_CODE LIKE '20_'
END
```

The output is:

```
LAST_NAME        FIRST_NAME  COURSE_NAME                      COURSE_CODE
---------        ----------  -----------                      -----------
BLACKWOOD        ROSEMARIE   WHAT'S NEW IN FOCUS              202
JONES            DIANE       FOCUS INTERNALS                  203
                             ADVANCED TECHNIQUES              201
```

Using Escape Characters for LIKE and UNLIKE

How to:

Specify an Escape Character for a Mask in an IF Phrase

Example:

Screening With an Escape Character in an IF Phrase

Reference:

Usage Notes for Escape Characters in IF Phrases

There may be times when you want to use the percent sign or underscore as a normal character in the mask, not as a wildcard character. You can indicate when you want to treat the percent sign or underscore as a normal character by preceding them with an escape character in the mask. This technique enables you to search for the percent sign or underscore character in the data.

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

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

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

where:

field

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

mask1, mask2

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

a, b ...

Are single characters that you identify as escape characters. Each mask can specify its own escape character or use the same character as other masks. If you embed the escape character in the mask, before a % or _, the % or _ character is treated as a literal, rather than as a wildcard. The single quotation marks are required if the mask contains blanks.

Reference: Usage Notes for Escape Characters in IF Phrases

- The escape character itself can be escaped, thus becoming a normal character in a string (for example, 'abc\%\\').
- The use of an escape character in front of any character other than %, _, and itself is ignored.
- Multiple escape characters can be used per LIKE phrase in an IF test.
- The escape character is only in effect when the ESCAPE syntax is included in the LIKE phrase.

Example: Screening With an Escape Character in an IF Phrase

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

```
TABLE FILE VIDEOTR2
PRINT CUSTID LASTNAME FIRSTNAME EMAI
IF EMAIL LIKE 'handy_%'
END
```

Because the underscore character functions as a wildcard character, this request returns two instances, only one of which contains the underscore character.

The output is:

```
CUSTID  LASTNAME          FIRSTNAME  EMAIL
------  --------          ---------  -----
0944    HANDLER           EVAN       handy_man@usa.com
0944    HANDLER           EVAN       handyman@usa.com
```

To retrieve only the instance that contains the underscore character, you must indicate that the underscore should be treated as a normal character, not a wildcard. The following request retrieves only the instance with the underscore character in the e-mail field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID LASTNAME FIRSTNAME EMAI
IF EMAIL LIKE 'handy\_%' ESCAPE '\'
END
```

The output is:

```
CUSTID  LASTNAME          FIRSTNAME  EMAIL
------  --------          ---------  -----
0944    HANDLER           EVAN       handy_man@usa.com
```

Assigning Column Numbers Only to Fields Displayed on Report Output

Column notation assigns a sequential column number to each column in the internal matrix created for a report request. You can use column notation in COMPUTE and RECAP commands to refer to these columns in your request.

Because column numbers refer to columns in the internal matrix, they are assigned after retrieval and aggregation are completed. Columns not actually displayed on the report output may exist in the internal matrix. For example, calculated values used in the request generate one or more columns in the internal matrix. Fields with the NOPRINT option take up a column in the internal matrix, and a reformatted field generates an additional column for the reformatted value. Certain RECAP calculations such as FORECAST or REGRESS generate multiple columns in the internal matrix.

BY fields are not assigned column numbers but, by default, every other column in the internal matrix is assigned a column number, which means that you have to account for all of the internally generated columns if you want to refer to the appropriate column value in your request. You can change this default column assignment behavior with the SET CNOTATION=PRINTONLY command, which assigns column numbers only to columns that display on the report output.

Syntax: How to Control the Creation of Column References

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

where:

`ALL`

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

`PRINTONLY`

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

`EXPLICIT`

Assigns column reference numbers to all columns used in a request, whether displayed or not. Details are described in the feature named Assigning Column Numbers to All Fields Used in a Request.

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

Using Column Notation in a Report Request

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

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

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

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

```
SET CNOTATION = ALL

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

The output is:

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

BY fields do not get a column reference, so the first column reference is for TRANSTOT with its original format, then the reformatted version. Next is QUANTITY with its original format and then the reformatted version. Last is the calculated value, PRODUCT, which represents TRANSTOT (F7.2) multiplied by TRANSTOT (D12.2).

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

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

```
SET CNOTATION = PRINTONLY

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

The output is:

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

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

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

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

```
SET CNOTATION=PRINTONLY
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
END
TABLE FILE LEDGER
SUM CUR_YR/F9.2 AS 'CURRENT,YEAR'
LAST_YR/F9.2 AS 'LAST,YEAR'

FOR ACCOUNT
1010 AS 'CASH ON HAND'                                 OVER
1020 AS 'DEMAND DEPOSITS'                              OVER
1030 AS 'TIME DEPOSITS'                                OVER
BAR                                                    OVER
RECAP TOTCASH/F9.2C= R1 + R2 + R3; AS 'TOTAL CASH' OVER
" "                                                    OVER
RECAP GROCASH(2)/F9.2C=100*TOTCASH(1)/TOTCASH(2) - 100;
AS 'CASH GROWTH(%)'
END
```

The output is:

```
                   CURRENT        LAST
                   YEAR           YEAR
                   -------        ----
CASH ON HAND       8784.00     7214.00
DEMAND DEPOSITS    4494.00     3482.00
TIME DEPOSITS      7961.00     6499.00
                 ---------   ---------
TOTAL CASH       21,239.00   17,195.00

CASH GROWTH(%)                   23.52
```

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

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

```
SET CNOTATION=PRINTONLY
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
NEXT_YR/I5C=1.13*CUR_YR + 222;
END
TABLE FILE LEDGER
SUM NEXT_YR/F9.2 CUR_YR/F9.2 LAST_YR/F9.2
FOR ACCOUNT
10$$ AS 'CASH'                          OVER
1100 AS 'ACCOUNTS RECEIVABLE'           OVER
1200 AS 'INVENTORY'                     OVER
BAR                                     OVER
RECAP ATOT(2,3)/I5C = R1 + R2 + R3;
AS 'ASSETS  ACTUAL'
END
```

The output is:

```
                          NEXT_YR     CUR_YR    LAST_YR
                          -------     ------    -------
CASH                     25991.00   21239.00   17195.00
ACCOUNTS RECEIVABLE      21941.00   18829.00   15954.00
INVENTORY                31522.00   27307.00   23329.00
                        ---------  ---------  ---------
ASSETS  ACTUAL                        67,375     56,478
```

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

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

```
SET CNOTATION=PRINTONLY
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
NEXT_YR/I5C=1.13*CUR_YR + 222;
END
TABLE FILE LEDGER
SUM NEXT_YR/F9.2 CUR_YR/F9.2 LAST_YR/F9.2
FOR ACCOUNT
10$$ AS 'TOTAL CASH' LABEL TOTCASH             OVER
" "                                            OVER
RECAP CHGCASH(1,2)/I5SC = TOTCASH(*) - TOTCASH(*+1); AS 'CHANGE IN CASH'
END
```

The output is:

```
                    NEXT_YR      CUR_YR    LAST_YR
                    -------      ------    -------
TOTAL CASH        25991.00    21239.00   17195.00

CHANGE IN CASH       4,752       4,044
```

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

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

```
SET CNOTATION=PRINTONLY
TABLE FILE REGION
SUM E_ACTUAL/F9.2 E_BUDGET/F9.2 W_ACTUAL/F9.2 W_BUDGET/F9.2
FOR ACCOUNT
3000 AS 'SALES'                         OVER
3100 AS 'COST'                          OVER
BAR                                     OVER
RECAP PROFIT/I5C = R1 - R2;             OVER
" "                                     OVER
RECAP EVAR(1)/I5C = E(3,1) - E(3,2);
AS 'EAST  VARIANCE'                     OVER
RECAP WVAR(3)/I5C = E(3,3) - E(3,4);
AS 'WEST  VARIANCE'
END
```

The output is:

```
                 E_ACTUAL   E_BUDGET   W_ACTUAL   W_BUDGET
                 --------   --------   --------   --------
SALES             6000.00    4934.00    7222.00    7056.00
COST              4650.00    3760.00    5697.00    5410.00
                ---------  ---------  ---------  ---------
PROFIT              1,350      1,174      1,525      1,646

EAST  VARIANCE        176
WEST  VARIANCE                             -121
```

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

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

```
SET CNOTATION = PRINTONLY
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
NEXT_YR/I5C=1.13*CUR_YR + 222;
END
TABLE FILE LEDGER
SUM NEXT_YR NOPRINT CUR_YR
COMPUTE AMT2/D6 = AMOUNT *2;
LAST_YR/D5   AMOUNT NEXT_YR
COMPUTE AMT3/D6   = AMOUNT*3;
COMPUTE AMT4/D6   = AMOUNT*4;
FOR ACCOUNT
10$$ AS 'CASH'                                   OVER
1100 AS 'ACCTS. REC.'                            OVER
1200 AS 'INVENTORY'                              OVER
BAR                                              OVER
RECAP ATOT/I8C = R1 + R2 + R3; AS 'TOTAL'        OVER
RECAP DIFF(2,10,2)/D8   = ATOT(*) - ATOT(*-1);
END
```

The output is:

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

Reference: Usage Notes for SET CNOTATION

- ❏ BY fields are not assigned column reference numbers.
- ❏ ACROSS fields do have column reference numbers and they will always show.
- ❏ Columns created internally can be seen in non-FML requests by creating a HOLD file from the request and examining the HOLD Master File that is created.

Assigning Column Numbers to All Fields Used in a Request

How to:

Control the Creation of Column Reference Numbers

Example:

Using Column Notation With NOPRINT in a non-FML Request

Using Cell Notation in an FML Request

Reference:

Usage Notes for Column Numbers

Column notation enables you to refer to columns in the internal matrix by number instead of field name when performing calculations on those values.

Certain reporting options create additional columns in the internal matrix that do not display on the report output. For example, when the request reformats a field, the original field and the reformatted field both become columns in the internal matrix. In this case, both versions of the field are assigned column numbers. Other reporting options can create several generated fields in the internal matrix to hold intermediate calculations. By default, you must take all of these additional columns into account when using column notation in a request.

The CNOTATION = EXPLICIT parameter assigns column numbers only to fields that are explicitly referenced in the request, eliminating the need to think about column notation references for generated fields in the matrix. This parameter also applies to FML requests that use column numbers, contiguous column notation in RECAP expressions, column addressing, relative column addresses, and cell notation.

CNOTATION has three possible values:

- ALL to create column notation for all fields. This is the default value.
- PRINTONLY to create column notation for fields displayed on the report output.
- EXPLICIT to create column notation for fields explicitly stated in the request whether displayed or not.

Syntax: How to Control the Creation of Column Reference Numbers

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

where:

`ALL`

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

`PRINTONLY`

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

`EXPLICIT`

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

Reference: Usage Notes for Column Numbers

- BY fields are not assigned column numbers.
- ACROSS columns are assigned column numbers.
- Calculated fields are assigned column numbers.
- Column numbers outside the range of the columns created in the request are allowed under the following circumstances (and are treated as containing the value zero):
  - When specified in a COMPUTE command issued after an ACROSS phrase.
  - In a cell reference in an FML RECAP command.

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

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

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

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

- TTOT2, which has the same value as TRANSTOT and displays on the report output.
- UNIT_COST1, which is calculated by dividing column1 by column2.
- UNIT_COST2, which is calculated by dividing column1 by QUANTITY.

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

In this request, only the setting CNOTATION=EXPLICIT gives the correct result. The following discussion clarifies the reason the EXPLICIT setting is required.

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

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

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

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

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

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

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

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

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

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

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

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

Example: Using Cell Notation in an FML Request

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

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

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

```
                     LAST_YR  NEXT_YR
                     -------  -------
TOTAL CASH            17,195   25,991

CHANGE FROM CURRENT   -4,044
```

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

```
                     LAST_YR  NEXT_YR
                     -------  -------
TOTAL CASH            17,195   25,991

CHANGE FROM CURRENT
```

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

```
                     LAST_YR  NEXT_YR
                     -------  -------
TOTAL CASH            17,195   25,991

CHANGE FROM CURRENT   -4,044    4,752
```

Controlling Underlining of ACROSS Objects

How to:

Control Underlining for ACROSS Objects

Example:

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

Removing Underlines for ACROSS Objects (SET ACROSSLINE=SKIP)

Replacing the Underline With a Blank LIne (SET ACROSSLINE=OFF)

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

Syntax: How to Control Underlining for ACROSS Objects

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

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

where:

`ON`

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

`OFF`

Replaces the underline with a blank line.

`SKIP`

Specifies no underline and no blank line.

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

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

The output is:

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

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

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

The output is:

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

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

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

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

```
                      Midwest    Northeast     Southeast         West
Product

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

Combinations of Summary Commands

How to:

Use Summary Commands

Reference:

Usage Notes for Combinations of Summary Commands

Example:

Using SUBTOTAL and RECOMPUTE in a Request

Using SUB-TOTAL With Multiple Summary Commands

Using Multiple Summary Commands With Prefix Operators

Propagation of Summary Commands With Field Lists

You can specify a different summary operation for each sort break (BY field). In prior releases, only one summary command (SUBTOTAL, SUB-TOTAL, SUMMARIZE, or RECOMPUTE) was supported in a request.

If you have multiple summary commands for the same sort field, the following message displays and the last summary command specified in the request is used:

```
(FOC36359)  MORE THAN 1 SUBTOTAL/SUB-TOTAL/RECOMPUTE/SUMMARIZE
```

There is more than one SUBTOTAL/SUB-TOTAL/RECOMPUTE/SUMMARIZE on the same key field which is not allowed. The last one specified will override the rest.

SUMMARIZE and SUB-TOTAL, which propagate their summary operations to higher level sort breaks, skip those fields at higher level sort breaks that have their own summary commands. The propagation of summary operations depends on whether prefix operator processing is used for summary lines. If prefix operators are:

❏ Not used on summary lines or if you issue the SET SUMMARYLINES=OLD command, prefix operator processing is not used for the request. In this case, if any summary command specifies a field list, only the fields specified on the summary line field lists are populated on the report.

- ❏ Used on summary lines or if you issue the SET SUMMARYLINES=NEW command, prefix operator processing is used for the request. In this case, SUB-TOTAL and SUMMARIZE propagate to:
  - ❏ All fields at higher level sort breaks that do not have their own summary command.
  - ❏ Fields not specified in the field list at higher level sort breaks that do have their own summary commands (columns that would have been empty). Note that this is the only technique that allows different fields at the same sort break to have different summary options.

Prefix operators on summary lines result in the same values whether the command is RECOMPUTE/SUMMARIZE or SUBTOTAL/SUB-TOTAL. For a computed field, the prefix operator is not applied, the value is recalculated using the expression in the COMPUTE command and the values from the summary line.

When you use different summary commands for different sort fields, the default grand total row inherits the summary command associated with the first sort field in the request. You can change the operation performed at the grand total level by using the ON TABLE phrase to specify a specific summary command.

Note: The grand total is considered the highest sort level. Therefore, although you can use the SUMMARIZE or SUB-TOTAL command at the grand total level, these commands apply only to the grand total and are not propagated to any other line on the report. On the grand total level SUMMARIZE operates as a RECOMPUTE command, and SUB-TOTAL operates as a SUBTOTAL command.

Syntax: How to Use Summary Commands

At a sort break, use the following syntax:

```
{BY|ON} breakfield [AS 'text1'] sumoption [MULTILINES]
        [pref.] [*|[field1 [[pref2.]field2 ...]]]
        [AS 'text2'] [WHEN expression;]
```

To replace the default grand total, use the following syntax

```
ON TABLE sumoption [pref.][field1 [[pref2.]field2 ...]] [AS 'text2']
```

where:

breakfield

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

sumoption

Can be one of the following: SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE.

'text1'

Is the column heading to use for the break field on the report output.

`MULTILINES`

Suppresses the printing of a summary line for every sort break that has only one detail line. Note that MULTILINES suppresses the summary line even if a prefix operator is used to specify a different operation for the summary line. MULTI-LINES is a synonym for MULTILINES.

`pref.`

Is a prefix operator. When specified without a field list, the prefix operator is applied to every numeric column in the report output and every numeric column is populated with values on the summary row.

`*`

Includes all display fields on the summary line. If a prefix operator is specified, it is applied to all fields. If the prefix operator is not supported with alphanumeric fields, alphanumeric fields are not included on the summary line.

`[`*`field1`*` [`*`field2`*` ... `*`fieldn`*`]]`

Produces the type of summary specified by *`sumoption`* for the listed fields. If no field names are listed, the summary is produced for every numeric column in the report output.

`pref. field1`` [`*`field2 ... fieldn`*`] [`*`pref2. fieldm ...`*`]`

The first prefix operator is applied to field1 through fieldn. The second prefix operator is applied to fieldm. Only the fields specified are populated with values on the summary row. Each prefix operator must be separated by a blank space from the following field name.

`'`*`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.

Example: Using SUBTOTAL and RECOMPUTE in a Request

In the following request, the first sort field specified is COPIES, which is associated with the RECOMPUTE command. Therefore, on the grand total line, the value of RATIO is correctly recomputed and the values of LISTPR and WHOLESALEPR are summed (because this is the default operation when the field is not calculated by a COMPUTE command).

```
TABLE FILE MOVIES
PRINT DIRECTOR LISTPR WHOLESALEPR
COMPUTE RATIO = LISTPR/WHOLESALEPR;
BY COPIES
BY RATING
WHERE COPIES LT 3
WHERE DIRECTOR EQ 'DISNEY W.' OR 'HITCHCOCK A.'
ON COPIES RECOMPUTE AS '*REC: '
ON RATING SUBTOTAL AS '*SUB:  '
END
```

The output is:

```
COPIES  RATING  DIRECTOR            LISTPR  WHOLESALEPR           RATIO
------  ------  --------            ------  -----------           -----
     1  NR      DISNEY W.            29.95        15.99            1.87

*SUB:   NR                           29.95        15.99            1.87
*REC:    1                           29.95        15.99            1.87

     2  NR      HITCHCOCK A.         19.98         9.00            2.22

*SUB:   NR                           19.98         9.00            2.22

        PG      HITCHCOCK A.         19.98         9.00            2.22
                HITCHCOCK A.         19.98         9.00            2.22

*SUB:   PG                           39.96        18.00            4.44
     2  PG13    HITCHCOCK A.         19.98         9.00            2.22

*SUB:   PG13                         19.98         9.00            2.22

        R       HITCHCOCK A.         19.98         9.00            2.22

*SUB:   R                            19.98         9.00            2.22
*REC:    2                           99.90        45.00            2.22


TOTAL                               129.85        60.99            2.13
```

If you reverse the BY fields, the grand total line sums the RATIO values as well as the LISTPR and WHOLESALEPR values because the SUBTOTAL command controls the grand total line:

```
TOTAL                                  129.85         60.99            12.97
```

You can change the operation performed at the grand total level by adding the following command to the request:

```
ON TABLE RECOMPUTE
```

The grand total line then displays the recomputed values:

```
TOTAL                                  129.85         60.99             2.13
```

Example: Using SUB-TOTAL With Multiple Summary Commands

In the following request, the SUB-TOTAL command propagates its operation to the DIRECTOR sort field (see the total line for HITCHCOCK, on which the RATIO values are subtotaled, not recomputed).

SUB-TOTAL is not propagated to the RATING sort field which has its own RECOMPUTE command, and for this sort field the RATIO value is recomputed.

The grand total line is recomputed because RECOMPUTE is performed on a higher level sort field than SUB-TOTAL.

```
TABLE FILE MOVIES
PRINT LISTPR WHOLESALEPR
COMPUTE RATIO = LISTPR/WHOLESALEPR;
BY DIRECTOR
BY RATING
BY COPIES
WHERE COPIES LT 3
WHERE DIRECTOR EQ 'HITCHCOCK A.'
ON COPIES SUB-TOTAL AS '*SUB: '
ON RATING RECOMPUTE AS '*REC:  '
END
```

The output is:

```
DIRECTOR           RATING  COPIES  LISTPR  WHOLESALEPR            RATIO
--------           ------  ------  ------  -----------            -----
HITCHCOCK A.       NR           2   19.98         9.00             2.22

*SUB:    2                          19.98         9.00             2.22
*REC:   NR                          19.98         9.00             2.22

                   PG           2   19.98         9.00             2.22
                                    19.98         9.00             2.22

*SUB:    2                          39.96        18.00             4.44
*REC:   PG                          39.96        18.00             2.22

                   PG13         2   19.98         9.00             2.22
*SUB:    2                          19.98         9.00             2.22
*REC:   PG13                        19.98         9.00             2.22

HITCHCOCK A.       R            2   19.98         9.00             2.22

*SUB:    2                          19.98         9.00             2.22
*REC:   R                           19.98         9.00             2.22
*TOTAL DIRECTOR HITCHCOCK A.        99.90        45.00            11.10


TOTAL                               99.90        45.00             2.22
```

Example: Using Multiple Summary Commands With Prefix Operators

The following request prints the average value of LISTPR and the recomputed value of RATIO on the lines associated with sort field RATING. The SUB-TOTAL command associated with sort field COPIES is propagated to all fields on the DIRECTOR sort field lines and to the WHOLESALEPR and RATIO1 columns associated with the RATING sort field. The grand total line is suppressed for this request.

```
TABLE FILE MOVIES
PRINT LISTPR WHOLESALEPR
COMPUTE RATIO/D6.2 = LISTPR/WHOLESALEPR;
COMPUTE RATIO1/D6.2 = LISTPR/WHOLESALEPR;
BY DIRECTOR
BY RATING
BY COPIES
WHERE COPIES LT 3
  WHERE DIRECTOR EQ 'KAZAN E.'
  ON RATING  RECOMPUTE  AVE. LISTPR  RATIO AS '*REC:  '
  ON COPIES  SUB-TOTAL                     AS '*SUB:  '
  ON TABLE NOTOTAL
END
```

On the output:

- ❑ The values of WHOLESALEPR and RATIO1 on the row labeled *REC are subtotals because of propagation of the SUB-TOTAL command to the fields not specified in the RECOMPUTE command.
- ❑ The LISTPR value is an average and the value of RATIO (which has the same definition as RATIO1) is recomputed because these two fields are specified in the RECOMPUTE command.
- ❑ The SUB-TOTAL command is propagated to the DIRECTOR row.

The output is:

```
DIRECTOR            RATING  COPIES  LISTPR  WHOLESALEPR   RATIO  RATIO1
--------            ------  ------  ------  -----------   -----  ------
KAZAN E.            NR           1   24.98        14.99    1.67    1.67

*SUB:     1                          24.98        14.99    1.67    1.67

                                 2   19.95         9.99    2.00    2.00

*SUB:     2                          19.95         9.99    2.00    2.00
*REC:   NR                           22.46        24.98     .90    3.66
*TOTAL DIRECTOR KAZAN E.             44.93        24.98    3.66    3.66
```

Example: Propagation of Summary Commands With Field Lists

In the following request, the RECOMPUTE command has a field list.

```
SET SUMMARYLINES = OLD
TABLE FILE MOVIES
PRINT LISTPR WHOLESALEPR
COMPUTE RATIO/D6.2 = LISTPR/WHOLESALEPR;
COMPUTE RATIO1/D6.2 = LISTPR/WHOLESALEPR;
BY DIRECTOR
BY RATING
BY COPIES
WHERE COPIES LT 3
  WHERE DIRECTOR EQ 'KAZAN E.'
  ON RATING RECOMPUTE LISTPR RATIO AS '*REC:   '
  ON COPIES SUB-TOTAL AS '*SUB:   '
END
```

With SUMMARYLINES=OLD, only those fields have values on the report output:

```
DIRECTOR           RATING  COPIES  LISTPR  WHOLESALEPR   RATIO  RATIO1
--------           ------  ------  ------  -----------   -----  ------
KAZAN E.           NR           1   24.98        14.99    1.67    1.67

*SUB:     1                         24.98                 1.67

                                2   19.95         9.99    2.00    2.00

*SUB:     2                         19.95                 2.00
*REC:   NR                          44.93                 1.80
*TOTAL DIRECTOR KAZAN E.            44.93                 3.66


TOTAL                               44.93                 1.80
```

With SUMMARYLINES=NEW, SUB-TOTAL propagates to all of the columns that would otherwise be unpopulated. The grand total line inherits the RECOMPUTE command for the fields listed in its field list, and the SUB-TOTAL command propagates to the other columns:

```
DIRECTOR            RATING  COPIES  LISTPR  WHOLESALEPR   RATIO  RATIO1
--------            ------  ------  ------  -----------   -----  ------
KAZAN E.            NR           1   24.98        14.99    1.67    1.67

*SUB:     1                          24.98        14.99    1.67    1.67

                                 2   19.95         9.99    2.00    2.00


*SUB:     2                          19.95         9.99    2.00    2.00
*REC:   NR                           44.93        24.98    1.80    3.66
*TOTAL DIRECTOR KAZAN E.             44.93        24.98    3.66    3.66



TOTAL                                44.93        24.98    1.80    3.66
```

Reference: Usage Notes for Combinations of Summary Commands

- Summary processing differs for summary commands that have prefix operators and summary lines that do not use prefix operators (see the *Creating Reports* manual for details). If any summary command uses prefix operators, the entire request uses prefix operator processing. If processing without prefix operators is initiated, but a subsequent field requires prefix operator processing, the following message is generated and processing halts:

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

  You can prevent this message by setting SUMMARYLINES=NEW to invoke prefix operator processing.

- SET SUMMARYLINES=EXPLICIT affects propagation of summary commands to the grand total line by making it consistent with the behavior for any sort break. Therefore, with this setting in effect, SUB-TOTAL and SUMMARIZE propagate to the grand total line but SUBTOTAL and RECOMPUTE do not.

Producing Summary Columns for Horizontal Sort Fields

How to:

Produce a Summary Operation on a Horizontal Sort Field

Example:

Using Summary Commands With ACROSS

Reference:

Usage Notes for Summaries on ACROSS Fields

The summary commands SUBTOTAL, SUB-TOTAL, SUMMARIZE, and RECOMPUTE can be used with horizontal sort breaks.

Syntax: How to Produce a Summary Operation on a Horizontal Sort Field

```
{ACROSS|ON} breakfield [AS 'text1'] sumoption [AS 'text2']
          [COLUMNS c1 [AND c2 ...]]
```

where:

breakfield

Is the ACROSS field whose for which you want to generate the summary option. The end of the values for the ACROSS field triggers the summary operation.

sumoption

Can be one of the following: SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE.

'text1'

Is the column heading to use for the break field on the report output.

'text2'

Is the text that prints on the top of the summary column.

COLUMNS *c1*, *c2* ...

Lists the specific ACROSS values that you want to display on the report output in the order in which you want them. This list of values cannot be specified in an ON phrase. If it is specified in an ACROSS phrase, it must be the last option specified in the ACROSS phrase.

Reference: Usage Notes for Summaries on ACROSS Fields

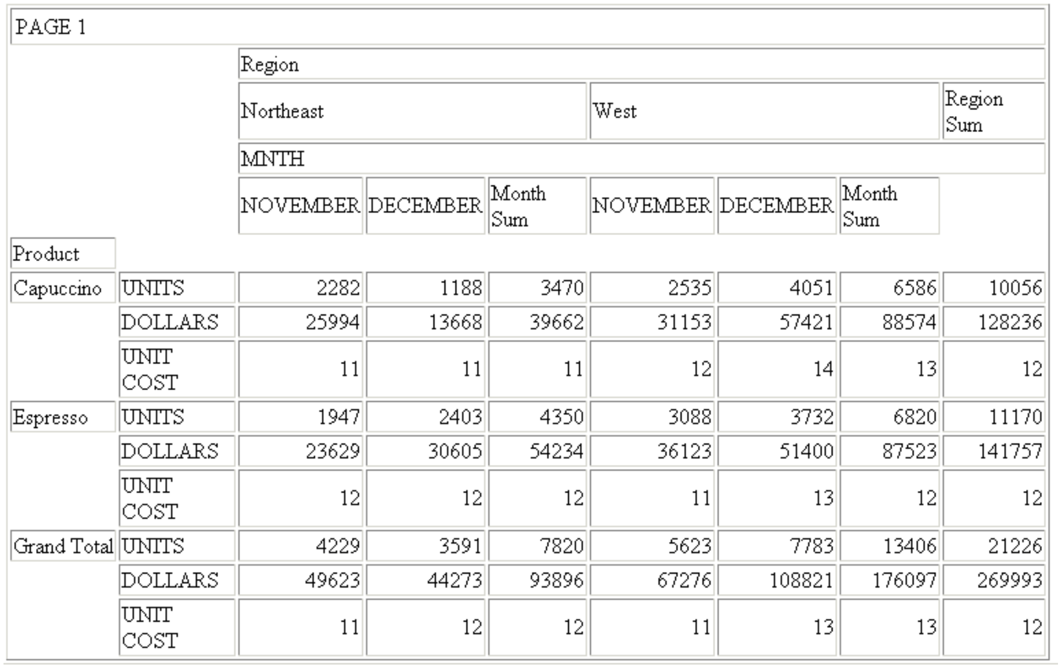
- ❑ SUMMARIZE and SUB-TOTAL operate on the ACROSS field for which they are specified and for all higher level ACROSS fields. They do not operate on BY fields. SUBTOTAL and RECOMPUTE operate only on the ACROSS field for which they are specified.
- ❑ SUMMARIZE and SUB-TOTAL commands specified for a BY field operate on that BY and all higher level BY fields. They do not operate on ACROSS fields.
- ❑ ROW-TOTAL, ACROSS-TOTAL, SUBTOTAL, and SUB-TOTAL sum the values in the columns. Unlike SUMMARIZE and RECOMPUTE, they do not reapply calculations other than sums.
- ❑ Summary commands specified in an ON TABLE phrase operate on columns, not rows.

Example: Using Summary Commands With ACROSS

The following request sums units and dollars and calculates the unit cost by product and across region and month. The ACROSS MNTH RECOMPUTE command creates totals of units and dollars, and recomputes the calculated value for the selected months within regions. The ACROSS REGION RECOMPUTE command does the same for the selected regions. The ON TABLE SUMMARIZE command creates summary rows. It has no effect on columns:

```
DEFINE FILE GGSALES
MNTH/MTr    = DATE;
END
TABLE FILE GGSALES
SUM
 UNITS/I5 AS 'UNITS'                       OVER
 DOLLARS/I6 AS 'DOLLARS'                   OVER
 COMPUTE DOLLPER/I6 = DOLLARS/UNITS; AS 'UNIT COST'
BY PRODUCT
ACROSS REGION RECOMPUTE AS 'Region Sum' COLUMNS 'Northeast' AND 'West'
ACROSS MNTH   RECOMPUTE AS 'Month Sum' COLUMNS 'November' AND 'December'
WHERE DATE FROM '19971101' TO '19971231';
WHERE PRODUCT EQ 'Capuccino' OR 'Espresso';
ON TABLE SUMMARIZE AS 'Grand Total'
ON TABLE HOLD FORMAT HTML
END
```

The output is:



PAGE 1								
		Region						
		Northeast			West			Region Sum
		MNTH						
		NOVEMBER	DECEMBER	Month Sum	NOVEMBER	DECEMBER	Month Sum	
Product								
Capuccino	UNITS	2282	1188	3470	2535	4051	6586	10056
	DOLLARS	25994	13668	39662	31153	57421	88574	128236
	UNIT COST	11	11	11	12	14	13	12
Espresso	UNITS	1947	2403	4350	3088	3732	6820	11170
	DOLLARS	23629	30605	54234	36123	51400	87523	141757
	UNIT COST	12	12	12	11	13	12	12
Grand Total	UNITS	4229	3591	7820	5623	7783	13406	21226
	DOLLARS	49623	44273	93896	67276	108821	176097	269993
	UNIT COST	11	12	12	11	13	13	12

Joining to a Fixed Format Sequential File Based on Multiple Fields

How to:

Join to a Fixed Format Sequential File Based on Multiple Fields

Example:

Joining to a Fixed Format Sequential Data Source Using Two Join Fields

Reference:

Usage Notes for Multi-Field Joins TO Fixed Format Sequential Files

A fixed format sequential data source can be the host or cross-referenced file in a join based on multiple fields. In a join TO a fixed format sequential file, both files that participate in the join must be sorted in ascending order of the concatenation of join fields and the sequential file must consist of a single segment.

Syntax: How to Join to a Fixed Format Sequential File Based on Multiple Fields

```
JOIN field1 AND field2 [AND field3 [AND field4]] IN file1
 TO {ALL|MULTIPLE|UNIQUE} field1a AND field2a [AND field3a [AND field4a]]
 IN file2 AS joinname
END
```

where:

file1

Is the host file.

file2

Is a single segment fixed format sequential file to be used as the cross referenced file.

field1, ... field4

Are up to four fields in the host file.

field1a, ... field4a

Are fields in the cross referenced file. Each join field must share formats and values with its corresponding field in the host file. The number of join fields in the cross referenced file must be the same as the number of join fields in the host file. The file must be sorted by the concatenation of the join fields.

joinname

Is a name assigned to the join.

ALL|MULTIPLE

Specifies a one-to-many join. All matching records in the cross referenced file will be retrieved.

UNIQUE

Specifies a one-to-one join. At most one matching record in the cross referenced file will be retrieved.

END

Is required to terminate the JOIN command if it spans multiple lines.

Reference: Usage Notes for Multi-Field Joins TO Fixed Format Sequential Files

- Both files participating in the JOIN must be in ascending sequence of the concatenation of join fields.
- A many-to-many JOIN is not supported.
- A fixed format sequential data source used as the cross-referenced file in the join must consist of a single segment.

Example: Joining to a Fixed Format Sequential Data Source Using Two Join Fields

The following request creates a fixed format sequential file named FIXSALE from the GGSALES data source. This file is sorted by store code and state:

```
SET ASNAMES = ON
TABLE FILE GGSALES
SUM PCD PRODUCT CATEGORY UNITS DOLLARS
BY STCD BY ST
ON TABLE HOLD AS FIXSALE FORMAT ALPHA
END
```

The following JOIN command joins the GGSTORES data source to the FIXSALE sequential file using two join fields. Both data sources are sorted on these join fields, store code and state:

```
JOIN STORE_CODE AND STATE IN GGSTORES
TO STCD AND ST IN FIXSALE AS JOIN1
END
```

The following request reports from the joined data source. The store name and zip code fields are from the GGSTORES data source and the UNITS and DOLLARS fields are from the FIXSALE file:

```
TABLE FILE GGSTORES
PRINT STORE_NAME ZIP UNITS DOLLARS
BY STORE_CODE BY ST
END
```

The output is:

```
Store ID  ST  Store Name                Zip Code     UNITS   DOLLARS
--------  --  ----------                --------     -----   -------
R1019     TX  GOTHAM GRINDS #1019       77201       299737   3714978
R1020     IL  GOTHAM GRINDS #1020       60601       307581   3924401
R1040     CA  GOTHAM GRINDS #1040       90001       298070   3772014
R1041     GA  GOTHAM GRINDS #1041       30314       330283   4100107
R1044     MA  GOTHAM GRINDS #1044       02101       301909   3707986
R1088     TN  GOTHAM GRINDS #1088       38104       294647   3687057
R1100     CT  GOTHAM GRINDS #1100       06901       302440   3782049
R1109     NY  GOTHAM GRINDS #1109       10001       312326   3902275
R1200     FL  GOTHAM GRINDS #1200       32853       310302   3923215
R1244     CA  GOTHAM GRINDS #1244       94132       312500   3870258
R1248     WA  GOTHAM GRINDS #1248       98101       321469   4010685
R1250     MO  GOTHAM GRINDS #1250       48880       297727   3761286
```

Ranking Sort Field Values With the RNK. Prefix Operator

How to:

Calculate Ranks Using the RNK. Prefix Operator

Example:

Ranking Within Sort Groups

Using RNK. in a WHERE TOTAL Test

Using RNK. in a COMPUTE Command

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

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

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

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

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

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

where:

field

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

Example: Ranking Within Sort Groups

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

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

The output is:

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

Example: Using RNK. in a WHERE TOTAL Test

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

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

The output is:

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

Example: Using RNK. in a COMPUTE Command

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

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

The output is:

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

Displaying a Row for Data Excluded by a Sort Phrase (PLUS OTHERS)

How to:

Display Data Excluded by a Sort Phrase

Example:

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

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

Reference:

Usage Notes for PLUS OTHERS

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

Syntax: How to Display Data Excluded by a Sort Phrase

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

where:

`LOWEST`

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

`HIGHEST`

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

`n`

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

`srtfield`

Is the name of the sort field.

`text`

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

`othertext`

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

`m1`

Is the incremental value between sort field groups.

`n2`

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

`m3`

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

`n4`

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

Reference: Usage Notes for PLUS OTHERS

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

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

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

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

The output is:

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

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

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

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

The output is:

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

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

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

The output is:

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

SET EMPTYREPORT=ANSI

How to:

Produce ANSI-Compliant Reports With No Records

Example:

Counting Fields With SET EMPTYREPORT=ANSI

Summing Fields With EMPTYREPORT=ANSI

Setting EMPTYREPORT=ON

Printing Fields and Setting EMPTYREPORT=ANSI in the Request

The EMPTYREPORT=ANSI setting introduces ANSI-compliant handling of reports with zero records, producing a one-line report containing only the specified missing data character. If the request is a COUNT operation, however, a zero is displayed for the missing values.

Syntax: How to Produce ANSI-Compliant Reports With No Records

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

where:

`ANSI`

Produces a single-line report and displays the missing data character or a zero if a COUNT is requested. in each case, &RECORDS will be 0, and &LINES will be 1.

If the SQL Translator is invoked, ANSI automatically replaces OFF as the default setting for EMPTYREPORT.

`ON`

Produces an empty report (column headings with no content). This was the default behavior in prior releases.

`OFF`

Produces no report output. OFF is the default value except for SQL Translator requests. When the SQL Translator is invoked, ANSI replaces OFF as the default setting for the EMPTYREPORT parameter, so the results are the same as for the ANSI setting.

The command can also be issued from within a request:

```
ON TABLE SET EMPTYREPORT ANSI
```

Example: Counting Fields With SET EMPTYREPORT=ANSI

```
SET EMPTYREPORT=ANSI
TABLE FILE EMPLOYEE
COUNT CURR_SAL AND DEPARTMENT
WHERE LAST_NAME EQ 'PATRICK'
END
```

The output shows no records and one line, with a zero displayed for CURR_SAL and a period for Department:

```
NUMBER OF RECORDS IN TABLE=        0  LINES=      1
PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

CURR_SAL  DEPARTMENT
COUNT     COUNT
--------  ----------
       0           0
```

Example: Summing Fields With EMPTYREPORT=ANSI

```
SET EMPTYREPORT=ANSI
TABLE FILE EMPLOYEE
SUM CURR_SAL
IF LAST_NAME EQ 'PATRICK'
END
```

Here is the output. Note that with SUM, the default missing data character (.) is displayed for CURR_SAL.

```
>
 NUMBER OF RECORDS IN TABLE=        0  LINES=      1
  PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

CURR_SAL
--------
       .
```

Example: Setting EMPTYREPORT=ON

```
SET EMPTYREPORT=ON
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND DEPARTMENT
IF LAST_NAME EQ 'PATRICK'
END
```

With EMPTYREPORT=ON, there are no lines in the report.

```
>
 NUMBER OF RECORDS IN TABLE=        0  LINES=      0
  PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

CURR_SAL  DEPARTMENT
--------  ----------
```

Example: Printing Fields and Setting EMPTYREPORT=ANSI in the Request

With PRINT you get one line, with the default missing data character (.) displayed.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND DEPARTMENT
IF DEPARTMENT EQ 'PATRICK'
ON TABLE SET EMPTYREPORT ANSI
END
```

The output is:

```
NUMBER OF RECORDS IN TABLE=         0  LINES=      1
PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

CURR_SAL  DEPARTMENT
--------  ----------
       .  .
```

Preserving Filter Definitions During Join Parsing

How to:

Preserve Filter Definitions With KEEPFILTERS

Example:

Preserving Filters With KEEPFILTERS

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

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

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

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

Syntax: How to Preserve Filter Definitions With KEEPFILTERS

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

where:

OFF

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

ON

Preserves filters across joins.

Example: Preserving Filters With KEEPFILTERS

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

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

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

```
? FILTER

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

Next the filter is activated:

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

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

```
? FILTER

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

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

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

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

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

PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

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

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

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

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

```
? FILTER

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

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

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

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

```
? FILTER

SET FILE     FILTER NAME DESCRIPTION
--- -------- ----------- ---------------------------------
*   VIDEOTRK UNITPR
*   VIDEOTRK YEARS1
```

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

```
JOIN CLEAR J1
? FILTER

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

SET SUMMARYLINES=EXPLICIT

How to:

Prevent Propagation of SUBTOTAL and RECOMPUTE to the Grand Total

Reference:

Usage Notes for SET SUMMARYLINES=EXPLICIT

Example:

Using SET SUMMARYLINES With SUBTOTAL

Using COLUMN-TOTAL With SET SUMMARYLINES=EXPLICIT

The SET SUMMARYLINES=EXPLICIT command makes the processing of SUBTOTAL, SUB-TOTAL, SUMMARIZE, and RECOMPUTE on the grand total line consistent with how they work for sort field breaks.

When SUBTOTAL and RECOMPUTE are used at a sort break level, they do not propagate to other sort breaks. SUB-TOTAL and SUMMARIZE propagate to all higher level sort breaks.

The grand total can be considered the highest level sort field in a request. However, by default, all of the summary options, not just SUB-TOTAL and SUMMARIZE, propagate to the grand total level.

The SET SUMMARYLINES=EXPLICIT prevents the propagation of SUBTOTAL and RECOMPUTE to the grand total. In addition, if all summary commands in the request specify field lists, only the specified fields are aggregated and displayed on the grand total line.

When SUBTOTAL and RECOMPUTE are the only summary commands used in the request, a grand total line is produced only if it is explicitly specified in the request using the ON TABLE SUBTOTAL/SUB-TOTAL/RECOMPUTE/SUMMARIZE phrase. If the ON TABLE phrase specifies a field list, only those fields are aggregated and displayed.

Note that you can always suppress the grand total line using the ON TABLE NOTOTAL command in the request.

Syntax: How to Prevent Propagation of SUBTOTAL and RECOMPUTE to the Grand Total

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

where:

`OLD`

Propagates all summary operations to the grand total line. Does not allow mixing of fields with and without prefix operators on a summary command when the first field does not have an associated prefix operator. All fields listed in any summary command are populated on all summary lines. OLD is the default value.

`NEW`

Propagates all summary operations to the grand total line. Uses prefix operator processing for all summary commands (all summary fields without prefix operators are processed as though they had a SUM. operator). Fields listed in a summary command are populated only on summary lines created by that summary command and on summary lines created by propagation of that summary command. Supports display of alphanumeric fields on summary lines.

`EXPLICIT`

Does not propagate SUBTOTAL and RECOMPUTE to the grand total line. Uses prefix operator processing for all summary commands (all summary fields without prefix operators are processed as though they had a SUM. operator). Fields listed in a summary command are populated only on summary lines created by that summary command and on summary lines created by propagation of that summary command. Supports display of alphanumeric fields on summary lines.

Note: This command is not supported in a request using the ON TABLE SET syntax.

Reference: Usage Notes for SET SUMMARYLINES=EXPLICIT

If COLUMN-TOTAL is specified in the request, all numeric fields are totaled on the grand total line unless the COLUMN-TOTAL phrase lists specific fields. If the COLUMN-TOTAL phrase lists specific fields, those fields and any fields propagated by SUB-TOTAL or SUMMARIZE commands are totaled.

Example: Using SET SUMMARYLINES With SUBTOTAL

The following request using the MOVIES data source has a sort break for CATEGORY that subtotals the COPIES field and a sort break for RATING that subtotals the LISTPR field:

```
SET SUMMARYLINES=OLD
TABLE FILE MOVIES
SUM COPIES LISTPR WHOLESALEPR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN'
WHERE RATING   EQ 'G'
ON RATING SUBTOTAL COPIES
ON CATEGORY SUBTOTAL LISTPR
END
```

Running the request with SUMMARYLINES=OLD subtotals both COPIES and LISTPR at every sort break and propagates them to the grand total line:

```
RATING  CATEGORY  COPIES  LISTPR  WHOLESALEPR
------  --------  ------  ------  -----------
G       CHILDREN       7  101.89        54.49

*TOTAL CHILDREN        7  101.89
*TOTAL G               7  101.89


TOTAL                  7  101.89
```

Running the request with SUMMARYLINES=NEW subtotals COPIES only for the RATING sort break and subtotals LISTPR only for the CATEGORY sort break but propagates both to the grand total line:

```
RATING  CATEGORY  COPIES  LISTPR  WHOLESALEPR
------  --------  ------  ------  -----------
G       CHILDREN       7  101.89        54.49

*TOTAL CHILDREN           101.89
*TOTAL G               7


TOTAL                  7  101.89
```

Running the request with SUMMARYLINES=EXPLICIT subtotals COPIES only for the RATING sort break and subtotals LISTPR only for the CATEGORY sort break. It does not produce a grand total line:

```
RATING  CATEGORY  COPIES  LISTPR  WHOLESALEPR
------  --------  ------  ------  -----------
G       CHILDREN       7  101.89        54.49

*TOTAL CHILDREN           101.89
*TOTAL G               7
```

Adding the phrase ON TABLE SUBTOTAL WHOLESALEPR with SUMMARYLINES=EXPLICIT produces a grand total line with the WHOLESALEPR field subtotaled:

```
RATING  CATEGORY  COPIES  LISTPR  WHOLESALEPR
------  --------  ------  ------  -----------
G       CHILDREN       7  101.89        54.49

*TOTAL CHILDREN           101.89
*TOTAL G               7


TOTAL                                   54.49
```

Example: Using COLUMN-TOTAL With SET SUMMARYLINES=EXPLICIT

The following request using the MOVIES data source has a sort break for CATEGORY for which subtotals the COPIES field and a sort break for RATING that subtotals the LISTPR field. It also has an ON TABLE COLUMN-TOTAL phrase:

```
SET SUMMARYLINES=EXPLICIT
TABLE FILE MOVIES
SUM COPIES LISTPR WHOLESALEPR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN'
WHERE RATING   EQ 'G'
ON RATING SUBTOTAL COPIES
ON CATEGORY SUBTOTAL LISTPR
ON TABLE COLUMN-TOTAL
END
```

The grand total line displays a column total for all numeric columns because of the ON TABLE COLUMN-TOTAL phrase:

```
RATING  CATEGORY  COPIES  LISTPR  WHOLESALEPR
------  --------  ------  ------  -----------
G       CHILDREN       7  101.89        54.49

*TOTAL CHILDREN           101.89
*TOTAL G               7


TOTAL                  7  101.89        54.49
```

The following request has an ON TABLE SUBTOTAL WHOLESALEPR command. It also has an ON TABLE COLUMN-TOTAL phrase:

```
SET SUMMARYLINES=EXPLICIT
TABLE FILE MOVIES
SUM COPIES LISTPR WHOLESALEPR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN'
WHERE RATING   EQ 'G'
ON RATING SUBTOTAL COPIES
ON CATEGORY SUBTOTAL LISTPR
ON TABLE SUBTOTAL WHOLESALEPR
ON TABLE COLUMN-TOTAL
END
```

The grand total line displays a column total only for the WHOLESALEPR column because of the ON TABLE SUBTOTAL command:

```
RATING  CATEGORY  COPIES  LISTPR  WHOLESALEPR
------  --------  ------  ------  -----------
G       CHILDREN       7  101.89        54.49

*TOTAL CHILDREN           101.89
*TOTAL G               7


TOTAL                                   54.49
```

Using SUB-TOTAL instead of SUBTOTAL causes COPIES and LISTPR to be aggregated on the grand total line. WHOLESALEPR is totaled because it is listed in the COLUMN-TOTAL phrase. The subtotal for LISTPR propagates to the RATING sort break as well as to the grand total:

```
SET SUMMARYLINES=EXPLICIT
TABLE FILE MOVIES
SUM COPIES LISTPR WHOLESALEPR
BY RATING
BY CATEGORY
WHERE CATEGORY EQ 'CHILDREN'
WHERE RATING   EQ 'G'
ON RATING SUB-TOTAL COPIES
ON CATEGORY SUB-TOTAL LISTPR
ON TABLE COLUMN-TOTAL WHOLESALEPR
END
```

The output is:

```
RATING  CATEGORY  COPIES  LISTPR  WHOLESALEPR
------  --------  ------  ------  -----------
G       CHILDREN       7  101.89        54.49

*TOTAL CHILDREN           101.89
*TOTAL G               7  101.89


TOTAL                  7  101.89        54.49
```

ON TABLE SET BYPANEL

The SET BYPANEL command, which controls whether vertical sort (BY) fields are repeated on each panel of a report, is now supported in a TABLE request.

Syntax: How to Control Repetition of BY Fields on Report Panels in a Request

```
ON TABLE SET BYPANEL = option
END
```

where:

`option`

Is one of the following:

`ON` displays all BY fields specified in the report request on each panel, and prevents column splitting.

`OFF` displays BY fields on the first panel only. Column splitting is permitted. OFF is the default value.

`0` displays BY fields only on the first panel. This prevents column splitting.

`n` is the number of BY fields to be displayed; *n* is less than or equal to the total number of BY fields specified in the request, from the major sort (first BY field) down. This prevents column splitting.

Column splitting occurs when a report column is too large to fit on the defined panel. By default, the column is split, displaying as many characters as possible, and the remaining characters continue on the next panel.

Left Outer Join Support

How to:

Specify an Inner or Left Outer Equijoin Between Real Fields

Specify an Inner or Left Outer DEFINE-based Equijoin

Specify an Inner or Left Outer Conditional JOIN

Create Data Sources Used in Inner and Left Outer Join Examples

Reference:

Relational Data Adapters and Optimization of LEFT OUTER/INNER JOINs

Usage Notes for Inner and Outer JOIN Command Syntax

Example:

Creating an Inner Join

Creating a Left Outer Join

Creating Two Inner Joins With a Multipath Structure

Creating DB2 Data Sources Used in Join Examples

Optimizing an Outer Join Between DB2 Data Sources

Optimizing Two Outer Joins Between DB2 Data Sources

When you join two data sources, some records in one of the files may lack corresponding records in the other file. When a report omits records that are not in both files, the join is called an inner join. When a report displays all matching records, plus all records from the host file that lack corresponding cross-referenced records, the join is called a left outer join.

The SET ALL command globally determines how all joins are implemented. If the SET ALL=ON command is issued, all joins are treated as outer joins. With SET ALL=OFF, the default, all joins are treated as inner joins.

Each JOIN command can specify explicitly which type of join to perform, locally overruling the global setting. This syntax is supported for FOCUS, XFOCUS, Relational, VSAM, IMS, and Adabas. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

Reference: Relational Data Adapters and Optimization of LEFT OUTER/INNER JOINs

The relational data adapters can optimize (translate to a single SQL SELECT statement processed by the relational engine) outer joins and combinations of joins under the following conditions:

- ❏ The OPTIMIZATION setting must not be OFF.
- ❏ The data adapter setting SQLJOIN OUTER must be set to ON. ON is the default value for all relational adapters except DB2.
- ❏ The data sources that participate in the JOIN must be for the same relational engine on the same server. For DB2, they must reside in the same subsystem.
- ❏ For conditional joins, the join expressions must be optimizable.
- ❏ If the tables do not share a single retrieval path in the joined structure, the setting SET MULTIPATH=COMPOUND must be in effect. Without this setting, multiple path joins are not optimized.

A left outer join in the root path (path leading to the root segment) of an inner join is converted to an inner join and the following message is generated:

```
(FOC32456) ANSI-COMPLIANCE ENFORCED BY CHANGING OUTER JOIN TO INNER
```

A left outer join in the root-path of an inner join has been changed into an inner join.

Aggregation commands such as SUM or COUNT cause the RDBMS to duplicate data. This prevents the join from being optimized. However, in some cases the data adapter can eliminate duplicate rows before passing the data back to FOCUS. It does this by adding an ORDER BY clause on all primary key columns to the SELECT statement it passes to the RDBMS for each table so that it can compare the returned data from all of the tables. This is called data adapter managed native join optimization.

In order to invoke this type of optimization, primary keys must exist (KEYS>0 in the Access File) for all segments except for the lowest level segment referenced in a report request that includes:

- ❏ Aggregation operators such as SUM or COUNT, when the OPTIMIZATION setting is ON or FOCUS.
- ❏ Multiple FST. or LST. operators on a segment that could return more than one record per sort break, regardless of the OPTIMIZATION setting.

In these situations, data adapter native join optimization is invoked regardless of the MULTIPATH setting.

Syntax: How to Specify an Inner or Left Outer Equijoin Between Real Fields

```
JOIN [LEFT_OUTER|INNER] hfld1 [AND hfld2 ...] IN hostfile [TAG tag1]
     TO [UNIQUE|MULTIPLE|ALL] crfield
     [AND crfld2 ...] IN crfile [TAG tag2] [AS joinname]
END
```

where:

`LEFT OUTER`

Specifies a left outer join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

`INNER`

Specifies an inner join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

`hfld1`

Is the name of a field in the host file containing values shared with a field in the cross-referenced file. This field is called the host field.

`hfld2...`

Can be an additional field in the host file, with the caveats noted below. The phrase beginning with AND is required when specifying multiple fields.

- ❏ When you are joining two FOCUS data sources you can specify up to four alphanumeric fields in the host file that, if concatenated, contain values shared with the cross-referenced file. You may not specify more than one field in the cross-referenced file when the suffix of the file is FOC. For example, assume the cross-referenced file contains a phone number field with an area code-prefix-exchange format. The host file has an area code field, a prefix field, and an exchange field. You can specify these three fields to join them to the phone number field in the cross-referenced file. The JOIN command treats the three fields as one. Other data sources do not have this restriction on the cross-referenced file.

- ❏ For data adapters that support multi-field and concatenated joins, you can specify up to 16 fields. See your data adapter documentation for specific information about supported join features. Note that FOCUS data sources do not support these joins.

`hostfile`

Is the name of the host file.

`tag1`

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in the host file.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

`TO [UNIQUE|MULTIPLE|ALL] crfld1`

Is the name of a field in the cross-referenced file containing values that match those of *hfld1* (or of concatenated host fields). This field is called the cross-referenced field.

Note: UNIQUE returns only one instance and, if there is no matching instance in the cross-referenced file, it supplies default values (blank for alphanumeric fields and zero for numeric fields).

Use the MULTIPLE parameter when *crfld1* may have multiple instances in common with one value in *hfld1*. Note that ALL is a synonym for MULTIPLE, and omitting this parameter entirely is a synonym for UNIQUE.

`crfld2...`

Is the name of a field in the cross-referenced file with values in common with *hfld2*.

Note: *crfld2* may be qualified. This field is only available for data adapters that support multi-field joins.

`crfile`

Is the name of the cross-referenced file.

`tag2`

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in cross-referenced files. In a recursive join structure, if no tag name is provided, all field names and aliases are prefixed with the first four characters of the join name.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

`joinname`

Is an optional name of up to eight characters that you may assign to the join structure. You must assign a unique name to a join structure if:

- ❏ You want to ensure that a subsequent JOIN command does not overwrite it.
- ❏ You want to clear it selectively later.
- ❏ The structure is recursive.

Note: If you do not assign a name to the join structure with the AS phrase, the name is assumed to be blank. A join without a name overwrites an existing join without a name.

`END`

Required when the JOIN command is longer than one line; it terminates the command.

For information about required conditions for cross-referenced fields and restrictions on group fields, see the *Creating Reports* manual.

Syntax: How to Specify an Inner or Left Outer DEFINE-based Equijoin

```
JOIN [LEFT_OUTER|INNER] deffld WITH host_field ...
     IN hostfile [TAG tag1]
     TO [UNIQUE|MULTIPLE|ALL] cr_field
     IN crfile [TAG tag2] [AS joinname]
END
```

where:

`LEFT OUTER`

Specifies a left outer join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

`INNER`

Specifies an inner join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

`deffld`

Is the name of a virtual field for the host file (the host field). The virtual field can be defined in the Master File or with a DEFINE command.

`host_field`

Is the name of any real field in the host segment with which you want to associate the virtual field. This association is required to locate the virtual field.

The WITH phrase is required unless the KEEPDEFINES parameter is set to ON and *deffld* was defined prior to issuing the JOIN command.

To determine which segment contains the virtual field, use the ? DEFINE query after issuing the DEFINE command.

`hostfile`

Is the name of the host file.

`tag1`

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in host files.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

`TO [UNIQUE|MULTIPLE|ALL] crfld1`

Is the name of a real field in the cross-referenced data source whose values match those of the virtual field. This must be a real field declared in the Master File.

Note: UNIQUE returns only one instance and, if there is no matching instance in the cross-referenced file, it supplies default values (blank for alphanumeric fields and zero for numeric fields).

Use the MULTIPLE parameter when *crfld1* may have multiple instances in common with one value in *hfld1*. Note that ALL is a synonym for MULTIPLE, and omitting this parameter entirely is a synonym for UNIQUE.

`crfile`

Is the name of the cross-referenced file.

`tag2`

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in cross-referenced files. In a recursive joined structure, if no tag name is provided, all field names and aliases are prefixed with the first four characters of the join name.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

`joinname`

Is an optional name of up to eight characters that you may assign to the joined structure. You must assign a unique name to a join structure if:

❑ You want to ensure that a subsequent JOIN command does not overwrite it.

❑ You want to clear it selectively later.

❑ The structure is recursive, and you do not specify tag names.

If you do not assign a name to the joined structure with the AS phrase, the name is assumed to be blank. A join without a name overwrites an existing join without a name.

`END`

Required when the JOIN command is longer than one line; terminates the command.

For information about required conditions for cross-referenced fields and restrictions on group fields, see the *Creating Reports* manual.

Syntax: How to Specify an Inner or Left Outer Conditional JOIN

```
JOIN [LEFT_OUTER|INNER] FILE hostfile AT hfld1 [TAG tag1]
     [WITH hfld2]
     TO {UNIQUE|MULTIPLE|ALL}
     FILE crfile AT crfld [TAG tag2] [AS joinname]
     [WHERE expression1;
     [WHERE expression2;
     ...]

END
```

where:

`LEFT OUTER`

Specifies a left outer join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

`INNER`

Specifies an inner join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

`hostfile`

Is the host Master File.

`AT`

Links the correct parent segment or host to the correct child or cross-referenced segment. The field values used as the AT parameter are not used to cause the link. They are simply used as segment references.

`hfld1`

Is the field name in the host Master File whose segment will be joined to the cross-referenced data source. The field name must be at the lowest level segment in its data source that is referenced.

`tag1`

Is the optional tag name that is used as a unique qualifier for fields and aliases in the host data source.

`hfld2`

Is a data source field with which to associate a DEFINE-based conditional JOIN. For a DEFINE-based conditional join, the KEEPDEFINES setting must be ON, and you must create the virtual fields before issuing the JOIN command.

`UNIQUE`

Specifies a one-to-one relationship between *from_file* and *to_file*. Note that ONE is a synonym for UNIQUE.

Note: Unique returns only one instance and, if there is no matching instance in the cross-referenced file, it supplies default values (blank for alphanumeric fields and zero for numeric fields).

`MULTIPLE|ALL`

Specifies a one-to-many relationship between *from_file* and *to_file*. Note that ALL is a synonym for MULTIPLE.

`crfile`

Is the cross-referenced Master File.

`crfld`

Is the join field name in the cross-referenced Master File. It can be any field in the segment.

`tag2`

Is the optional tag name that is used as a unique qualifier for fields and aliases in the cross-referenced data source.

`joinname`

Is the name associated with the joined structure.

`expression1, expression2`

Are any expressions that are acceptable in a DEFINE FILE command. All fields used in the expressions must lie on a single path.

Note: Single line JOIN syntax is not supported. The END command is required.

Reference: Usage Notes for Inner and Outer JOIN Command Syntax

- ❑ The SET ALL and SET CARTESIAN commands are ignored by the syntax.
- ❑ The ALL. parameter is not supported. If the ALL. parameter is used, the following message displays:

  ```
  (FOC32452) Use of ALL. with LEFT_OUTER/INNER not allowed
  ```

- ❑ If you define multiple joins, the resulting structure can be a single path or multi-path data source:
  - ❑ If the setting MULTIPATH=SIMPLE is in effect and the report is based on multiple paths, each of the individual joins is constructed separately without regard to the other joins, and the matching records from each of the separate paths displays on the report output. Therefore, the output can contain records that would have been omitted if only one of the joins was in effect.
  - ❑ If the setting MULTIPATH=COMPOUND is in effect with a multi-path report, or if the report displays data only from a single path, the report output displays only those records that satisfy all of the joins.

Procedure: How to Create Data Sources Used in Inner and Left Outer Join Examples

The following procedure creates three FOCUS data sources:

- ❑ EMPINFO, which contains the fields EMP_ID, LAST_NAME, FIRST_NAME, and CURR_JOBCODE from the EMPINFO segment of the EMPLOYEE data source.
- ❑ JOBINFO, which contains the JOBCODE and JOB_DESC fields from the JOBFILE data source.
- ❑ EDINFO, which contains the EMP_ID, COURSE_CODE, and COURSE_NAME fields from the EDUCFILE data source.

The procedure then adds an employee to EMPINFO named Fred Newman who has no matching record in the JOBINFO or EDINFO data sources.

```
TABLE FILE EMPLOYEE
SUM LAST_NAME FIRST_NAME CURR_JOBCODE
BY EMP_ID
ON TABLE HOLD AS EMPINFO FORMAT FOCUS INDEX EMP_ID CURR_JOBCODE
END
-RUN

TABLE FILE JOBFILE
SUM JOB_DESC
BY JOBCODE
ON TABLE HOLD AS JOBINFO FORMAT FOCUS INDEX JOBCODE
END
-RUN

TABLE FILE EDUCFILE
SUM COURSE_CODE COURSE_NAME
BY EMP_ID
ON TABLE HOLD AS EDINFO FORMAT FOCUS INDEX EMP_ID
END
-RUN

MODIFY FILE EMPINFO
FREEFORM EMP_ID LAST_NAME FIRST_NAME CURR_JOBCODE
MATCH EMP_ID
ON NOMATCH INCLUDE
ON MATCH REJECT
DATA
111111111, NEWMAN, FRED, C07,$
END
```

The following request prints the contents of EMPINFO. Note that Fred Newman has been added to the data source:

```
TABLE FILE EMPINFO
PRINT *
END
```

The output is:

```
EMP_ID     LAST_NAME        FIRST_NAME  CURR_JOBCODE
------     ---------        ----------  ------------
071382660  STEVENS          ALFRED      A07
112847612  SMITH            MARY        B14
117593129  JONES            DIANE       B03
119265415  SMITH            RICHARD     A01
119329144  BANNING          JOHN        A17
123764317  IRVING           JOAN        A15
126724188  ROMANS           ANTHONY     B04
219984371  MCCOY            JOHN        B02
326179357  BLACKWOOD        ROSEMARIE   B04
451123478  MCKNIGHT         ROGER       B02
543729165  GREENSPAN        MARY        A07
818692173  CROSS            BARBARA     A17
111111111  NEWMAN           FRED        C07
```

Example: Creating an Inner Join

The following JOIN command creates an inner join between the EMPINFO data source and the JOBINFO data source.

```
JOIN CLEAR *
JOIN INNER CURR_JOBCODE IN EMPINFO TO MULTIPLE JOBCODE IN JOBINFO AS J0
```

Note that the JOIN command specifies a multiple join. In a unique join, the cross-referenced segment is never considered missing, and all records from the host file display on the report output. Default values (blank for alphanumeric fields and zero for numeric fields) display if no actual data exists.

The following request displays fields from the joined structure:

```
TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME JOB_DESC
END
```

Fred Newman is omitted from the report output because his job code does not have a match in the JOBINFO data source:

```
LAST_NAME        FIRST_NAME  JOB_DESC
---------        ----------  --------
STEVENS          ALFRED      SECRETARY
SMITH            MARY        FILE QUALITY
JONES            DIANE       PROGRAMMER ANALYST
SMITH            RICHARD     PRODUCTION CLERK
BANNING          JOHN        DEPARTMENT MANAGER
IRVING           JOAN        ASSIST.MANAGER
ROMANS           ANTHONY     SYSTEMS ANALYST
MCCOY            JOHN        PROGRAMMER
BLACKWOOD        ROSEMARIE   SYSTEMS ANALYST
MCKNIGHT         ROGER       PROGRAMMER
GREENSPAN        MARY        SECRETARY
CROSS            BARBARA     DEPARTMENT MANAGER
```

Example: Creating a Left Outer Join

The following JOIN command creates a left outer join between the EMPINFO data source and the EDINFO data source:

```
JOIN CLEAR *
JOIN LEFT_OUTER EMP_ID IN EMPINFO TO MULTIPLE EMP_ID IN EDINFO AS J1
```

The following request displays fields from the joined structure:

```
TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME COURSE_NAME
END
```

All employee records display on the report output. The records for those employees with no matching records in the EDINFO data source display the missing data character (.) in the COURSE_NAME column. If the join were unique, blanks would display instead of the missing data character.

```
LAST_NAME        FIRST_NAME  COURSE_NAME
---------        ----------  -----------
STEVENS          ALFRED      FILE DESCRPT & MAINT
SMITH            MARY        BASIC REPORT PREP FOR PROG
JONES            DIANE       FOCUS INTERNALS
SMITH            RICHARD     BASIC RPT NON-DP MGRS
BANNING          JOHN        .
IRVING           JOAN        .
ROMANS           ANTHONY     .
MCCOY            JOHN        .
BLACKWOOD        ROSEMARIE   DECISION SUPPORT WORKSHOP
MCKNIGHT         ROGER       FILE DESCRPT & MAINT
GREENSPAN        MARY        .
CROSS            BARBARA     HOST LANGUAGE INTERFACE
NEWMAN           FRED        .
```

Example: Creating Two Inner Joins With a Multipath Structure

The following JOIN commands create an inner join between the EMPINFO and JOBINFO data sources and an inner join between the EMPINFO and EDINFO data sources:

```
JOIN CLEAR *
JOIN INNER CURR_JOBCODE IN EMPINFO TO MULTIPLE JOBCODE IN JOBINFO AS J0
JOIN INNER EMP_ID IN EMPINFO TO MULTIPLE EMP_ID IN EDINFO AS J1
```

The structure created by the two joins has two independent paths:

```
         SEG01
 01      S1
**************
*EMP_ID      **I
*CURR_JOBCODE**I
*LAST_NAME   **
*FIRST_NAME  **
*            **
***************
 **************
       I
       +-----------------+
       I                 I
       I SEG01           I SEG01
 02    I KM        03    I KM
..............    ..............
:EMP_ID      ::K  :JOBCODE      ::K
:COURSE_CODE ::   :JOB_DESC     ::
:COURSE_NAME ::   :             ::
:            ::   :             ::
:            ::   :             ::
:............::   :............::
 .............:    .............:
 JOINED  EDINFO    JOINED  JOBINFO
```

The following request displays fields from the joined structure:

```
SET MULTIPATH=SIMPLE
TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME IN 12 COURSE_NAME JOB_DESC
END
```

With MULTIPATH=SIMPLE, the independent paths create independent joins. All employee records accepted by either join display on the report output. Only Fred Newman (who has no matching record in either of the cross-referenced files) is omitted:

```
LAST_NAME  FIRST_NAME  COURSE_NAME                     JOB_DESC
---------  ----------  -----------                     --------
STEVENS    ALFRED      FILE DESCRPT & MAINT            SECRETARY
SMITH      MARY        BASIC REPORT PREP FOR PROG      FILE QUALITY
JONES      DIANE       FOCUS INTERNALS                 PROGRAMMER ANALYST
SMITH      RICHARD     BASIC RPT NON-DP MGRS           PRODUCTION CLERK
BANNING    JOHN        .                               DEPARTMENT MANAGER
IRVING     JOAN        .                               ASSIST.MANAGER
ROMANS     ANTHONY     .                               SYSTEMS ANALYST
MCCOY      JOHN        .                               PROGRAMMER
BLACKWOOD  ROSEMARIE   DECISION SUPPORT WORKSHOP       SYSTEMS ANALYST
MCKNIGHT   ROGER       FILE DESCRPT & MAINT            PROGRAMMER
GREENSPAN  MARY        .                               SECRETARY
CROSS      BARBARA     HOST LANGUAGE INTERFACE         DEPARTMENT MANAGER
```

With MULTIPATH=COMPOUND, only employees with matching records in both of the cross-referenced files display on the report output:

```
LAST_NAME  FIRST_NAME  COURSE_NAME                     JOB_DESC
---------  ----------  -----------                     --------
STEVENS    ALFRED      FILE DESCRPT & MAINT            SECRETARY
SMITH      MARY        BASIC REPORT PREP FOR PROG      FILE QUALITY
JONES      DIANE       FOCUS INTERNALS                 PROGRAMMER ANALYST
SMITH      RICHARD     BASIC RPT NON-DP MGRS           PRODUCTION CLERK
BLACKWOOD  ROSEMARIE   DECISION SUPPORT WORKSHOP       SYSTEMS ANALYST
MCKNIGHT   ROGER       FILE DESCRPT & MAINT            PROGRAMMER
CROSS      BARBARA     HOST LANGUAGE INTERFACE         DEPARTMENT MANAGER
```

Example: Creating DB2 Data Sources Used in Join Examples

The following procedure creates DB2 data sources to use in join examples. This assumes the Master Files and Access Files for EMPINFO, JOBINFO, and EDINFO already exist and that you have the authority to create tables:

```
TABLE FILE EMPLOYEE
SUM LAST_NAME FIRST_NAME CURR_JOBCODE
BY EMP_ID
ON TABLE HOLD AS EMPF FORMAT FOCUS
END
-RUN

TABLE FILE JOBFILE
SUM JOB_DESC
BY JOBCODE
ON TABLE HOLD AS JOB1 FORMAT ALPHA
END
-RUN

TABLE FILE EDUCFILE
SUM COURSE_CODE COURSE_NAME
BY EMP_ID
ON TABLE HOLD AS ED1 FORMAT ALPHA
END
-RUN

-*Create new record with no match in job or ed files
MODIFY FILE EMPF
FREEFORM EMP_ID LAST_NAME FIRST_NAME CURR_JOBCODE
MATCH EMP_ID
 ON NOMATCH INCLUDE
 ON MATCH REJECT
DATA
111111111, NEWMAN, FRED, C07,$
END
-RUN

TABLE FILE EMPF
PRINT *
ON TABLE HOLD AS EMP1 FORMAT ALPHA
END
-RUN

-*Create DB2 tables
CREATE FILE EMPINFO
-RUN
CREATE FILE JOBINFO
-RUN
```

```
CREATE FILE EDINFO
-RUN
-*Load DB2 tables
MODIFY FILE EMPINFO
FIXFORM FROM EMP1
DATA ON EMP1
END
-RUN
MODIFY FILE JOBINFO
FIXFORM FROM JOB1
DATA ON JOB1
END
-RUN
MODIFY FILE EDINFO
FIXFORM FROM ED1
DATA ON ED1
END
-RUN
```

Example: Optimizing an Outer Join Between DB2 Data Sources

The following request joins the EMPINFO table to the JOBINFO table. The SET SQLJOIN OUTER ON command is required for DB2, which has OFF as its default value:

```
ENGINE DB2 SET SQLJOIN OUTER ON
SET TRACEUSER=ON
SET TRACEOFF=ALL
SET TRACEON = STMTRACE//CLIENT
-RUN

JOIN CLEAR *
-RUN

JOIN
 LEFT_OUTER CURR_JOBCODE IN EMPINFO TO MULTIPLE JOBCODE IN JOBINFO AS J0
END

TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME IN 12 JOB_DESC
END
```

The single SQL SELECT statement for both tables shows that the join is optimized:

```
SELECT T1."LN",T1."FN",T1."CJC",T2."JCD",T2."JDS" FROM
(USERID1."EMPINFO" T1 LEFT OUTER JOIN USERID1."JOBINFO" T2 ON
T2."JCD" = T1."CJC") FOR FETCH ONLY;
```

Fred Newman is included although he has no matching record in the JOBINFO data source. His missing job description is indicated by the NODATA character (.):

```
LAST_NAME  FIRST_NAME  JOB_DESC
---------  ----------  --------
SMITH      RICHARD     PRODUCTION CLERK
GREENSPAN  MARY        SECRETARY
STEVENS    ALFRED      SECRETARY
IRVING     JOAN        ASSIST.MANAGER
BANNING    JOHN        DEPARTMENT MANAGER
CROSS      BARBARA     DEPARTMENT MANAGER
MCKNIGHT   ROGER       PROGRAMMER
MCCOY      JOHN        PROGRAMMER
JONES      DIANE       PROGRAMMER ANALYST
BLACKWOOD  ROSEMARIE   SYSTEMS ANALYST
ROMANS     ANTHONY     SYSTEMS ANALYST
SMITH      MARY        FILE QUALITY
NEWMAN     FRED        .
```

Example: Optimizing Two Outer Joins Between DB2 Data Sources

The following request joins the EMPINFO table to the JOBINFO table and the EDINFO table with the setting MULTIPATH=COMPOUND. The SET SQLJOIN OUTER ON command is required for DB2, which has OFF as its default value:

```
ENGINE DB2 SET SQLJOIN OUTER ON
SET MULTIPATH=COMPOUND
-RUN
SET TRACEUSER=ON
SET TRACEOFF=ALL
SET TRACEON = STMTRACE//CLIENT
-RUN

JOIN CLEAR *
-RUN

JOIN
  LEFT_OUTER CURR_JOBCODE IN EMPINFO TO MULTIPLE JOBCODE IN JOBINFO AS J0
  END
JOIN
  LEFT_OUTER EMP_ID IN EMPINFO  TO MULTIPLE EMP_ID IN EDINFO   AS J1
END

TABLE FILE EMPINFO
PRINT LAST_NAME  FIRST_NAME IN 12 COURSE_NAME JOB_DESC
END
```

The single SQL SELECT statement incorporating all tables shows that the join is optimized:

```
SELECT T1."EID",T1."LN",T1."FN",T1."CJC",T2."E01",T2."CNM",
T3."JCD",T3."JDS" FROM ( ( USERID1."EMPINFO" T1 LEFT OUTER JOIN
USERID1."EDINFO" T2 ON T2."E01" = T1."EID" ) LEFT OUTER JOIN
USERID1."JOBINFO" T3 ON T3."JCD" = T1."CJC" ) FOR FETCH ONLY;
```

All employees are included on the report output. Missing course names and job descriptions are indicated by the NODATA character (.):

```
LAST_NAME   FIRST_NAME   COURSE_NAME                      JOB_DESC
---------   ----------   -----------                      --------
SMITH       RICHARD      BASIC RPT NON-DP MGRS            PRODUCTION CLERK
GREENSPAN   MARY         .                                SECRETARY
STEVENS     ALFRED       FILE DESCRPT & MAINT             SECRETARY
IRVING      JOAN         .                                ASSIST.MANAGER
BANNING     JOHN         .                                DEPARTMENT MANAGER
CROSS       BARBARA      HOST LANGUAGE INTERFACE          DEPARTMENT MANAGER
MCKNIGHT    ROGER        FILE DESCRPT & MAINT             PROGRAMMER
MCCOY       JOHN         .                                PROGRAMMER
JONES       DIANE        FOCUS INTERNALS                  PROGRAMMER ANALYST
ROMANS      ANTHONY      .                                SYSTEMS ANALYST
BLACKWOOD   ROSEMARIE    DECISION SUPPORT WORKSHOP        SYSTEMS ANALYST
SMITH       MARY         BASIC REPORT PREP FOR PROG       FILE QUALITY
NEWMAN      FRED         .                                .
```

FMLCAP Function

How to:

Retrieve Captions in an FML Request Using the FMLCAP Function

Example:

Retrieving FML Hierarchy Captions Using FMLCAP

The FMLCAP function returns the caption value for each row in an FML hierarchy request. In order to retrieve caption values, the Master File must define an FML hierarchy and the request must use the GET CHILDREN, ADD, or WITH CHILDREN option to retrieve hierarchy data. If the FOR field in the request does not have a caption field defined, FMLCAP returns a blank string.

FMLCAP is supported for COMPUTE but is not recommended for use with DEFINE.

Syntax: How to Retrieve Captions in an FML Request Using the FMLCAP Function

```
FMLCAP(fieldname|'format')
```

where:

fieldname

Is the name of the caption field.

'format'

Is the format of the caption field enclosed in single quotation marks.

Example: Retrieving FML Hierarchy Captions Using FMLCAP

The following request retrieves and aggregates the FML hierarchy that starts with the parent value 2000. FMLCAP retrieves the captions, while the actual account numbers appear as the FOR values.

```
SET FORMULTIPLE = ON
TABLE FILE CENTSTMT
SUM ACTUAL_AMT
COMPUTE CAP1/A30= FMLCAP(GL_ACCOUNT_CAPTION);
FOR GL_ACCOUNT
2000 WITH CHILDREN 2 ADD
END
```

The output is:

```
                   Actual  CAP1
                   ------  ----
2000         313,611,852.  Gross Margin
  2100       187,087,470.  Sales Revenue
    2200      98,710,368.  Retail Sales
    2300      13,798,832.  Mail Order Sales
    2400      12,215,780.  Internet Sales
  2500       100,885,159.  Cost Of Goods Sold
    2600      54,877,250.  Variable Material Costs
    2700       6,176,900.  Direct Labor
    2800       3,107,742.  Fixed Costs
```

FMLFOR Function

How to:

Retrieve FML Tag Values

Example:

Retrieving FML Tag Values With FMLFOR

FMLFOR retrieves the tag value associated with each row in an FML request. If the FML row was generated as a sum of data records using the OR phrase, FMLFOR returns the first value specified in the list. If the OR phrase was generated by an FML Hierarchy ADD command, FMLFOR returns the tag value associated with the parent specified in the ADD command.

The FMLFOR function is supported for COMPUTE but not for DEFINE. Attempts to use it in a DEFINE result in blank values.

Syntax: How to Retrieve FML Tag Values

```
FMLFOR(outfield)
```

where:

outfield

Is name of the field that will contain the result, or the format of the output value enclosed in single quotation marks.

Example: Retrieving FML Tag Values With FMLFOR

```
SET FORMULTIPLE = ON
TABLE FILE LEDGER
SUM AMOUNT
COMPUTE RETURNEDFOR/A8 = FMLFOR('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
```

FMLLIST Function

How to:

Retrieve an FML Tag List

Example:

Retrieving an FML Tag List With FMLLIST

FMLLIST returns a string containing the complete tag list for each row in an FML request. If a row has a single tag value, that value is returned.

The FMLLIST function is supported for COMPUTE but not for DEFINE. Attempts to use it in a DEFINE result in blank values.

Syntax: How to Retrieve an FML Tag List

```
FMLLIST('A4096V')
```

where:

`'A4096V'`

Is the required argument.

Example: Retrieving an FML Tag List With FMLLIST

```
SET FORMULTIPLE=ON
TABLE FILE LEDGER
HEADING
"TEST OF FMLLIST"
" "
SUM AMOUNT
COMPUTE LIST1/A36 = FMLLIST('A4096V');
FOR ACCOUNT
'1010'                 OVER
'1020'                 OVER
'1030'                 OVER
BAR                    OVER
'1030' OR '1020' OR '1010'
END
```

The output is:

```
TEST OF FMLLIST

      AMOUNT  LIST1
      ------  -----
1010   8,784  1010
1020   4,494  1020
1030   7,961  1030
      ------  ------------------------------------
1010  21,239  1010 OR 1020 OR 1030
```

Changing Titles of Columns Created Using Prefix Operators

How to:

Rename a Report Column Created Using a Prefix Operator

Example:

Renaming a Report Column Created Using a Prefix Operator

Column titles for columns created by applying a prefix operator to a display or sort field can be renamed using an AS phrase in a report request.

Syntax: How to Rename a Report Column Created Using a Prefix Operator

```
operator.fieldname AS 'coltitle'
```

where:

`operator`

Is a valid prefix operator.

`fieldname`

Is a display or sort field in the request.

`coltitle`

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

Example: Renaming a Report Column Created Using a Prefix Operator

The following request sums wholesale prices of movies by category, counts the instances and sorts by the average wholesale price in descending order. The columns created with the AVE. and CNT. prefix operators are renamed using an AS phrase:

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

The output is:

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

Joining Comma-Delimited Files

How to:

Specify a JOIN

Reference:

Usage Notes for Comma-Delimited Files in a Join

Example:

Joining Two Comma-Delimited Files

In FOCUS 7.6.2, a comma-delimited file can be the host or cross-referenced file in a JOIN command. The setting PCOMMA = ON must be in effect. Therefore, alphanumeric fields must be enclosed in double quotation marks and all of the data must be on a single line, with the line-end character indicating the end of record. This format is consistent with a HOLD file created as format COM or COMT. Note that files created by HOLD FORMAT COM or HOLD FORMAT COMT are created using the PCOMMA ON format, and thus PCOMMA must be set to ON in order to issue a report request against these files.

In comma-delimited files, the cross-referenced field can be any field. However, both the host and cross-referenced files must be retrieved in ascending order of the concatenated join fields.

Syntax: How to Specify a JOIN

```
JOIN [LEFT_OUTER|INNER] field1 [AND field1a...] IN host [TAG tag1]
 TO [ALL|MULTIPLE|UNIQUE] field2 [AND field2a...] IN crfile [TAG tag2]
[AS joinname]
END
```

where:

field1

Is the name of a field in the host file containing values shared with a field in the cross-referenced file. This field is called the host field.

field1a...

Can be up to three additional fields in the host file. The phrase beginning with AND is required when specifying multiple fields.

INNER

Specifies an inner join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

LEFT_OUTER

Specifies a left outer join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

host

Is the name of the host file.

tag1

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in the host file.

The tag name for the host file must be the same in all the JOIN commands of a joined structure.

`[ALL|MULTIPLE|UNIQUE]`

Specifies whether a join is one-to-one or one-to-many.

UNIQUE returns only one instance and, if there is no matching instance in the cross-referenced file, it supplies default values (blank for alphanumeric fields and zero for numeric fields).

Use the MULTIPLE parameter when the cross-referenced field may have multiple instances in common with one value in the host file. Note that ALL is a synonym for MULTIPLE, and omitting this parameter entirely is a synonym for UNIQUE

Note: Many-to-many joins are not supported when the cross-referenced file is sequential.

`field2`

Is the name of a field in the cross-referenced file containing values that match those of field1 (or of concatenated host fields). This field is called the cross-referenced field and can be any field as long as the both data sources are in ascending order of the concatenation of join fields.

`field2a...`

Are the names of up to three fields in the cross-referenced file with values in common with their corresponding join fields in the host file.

`tag2`

Is a tag name of up to eight characters (usually the name of the Master File), which is used as a unique qualifier for fields and aliases in the cross-referenced file. In a recursive join structure, if no tag name is provided, all field names and aliases are prefixed with the first four characters of the join name.

`joinname`

Is an optional name of up to eight characters that you may assign to the join structure. You must assign a unique name to a join structure if:

- You want to ensure that a subsequent JOIN command does not overwrite it.
- You want to clear it selectively later.
- The structure is recursive.

Note: If you do not assign a name to the join structure with the AS phrase, the name is assumed to be blank. A join without a name overwrites an existing join without a name.

`END`

Is required when the JOIN command is longer than one line; it terminates the command.

Reference: Usage Notes for Comma-Delimited Files in a Join

- Both files in the join must be in ascending order of the concatenation of join fields. If the files are not in the same sort order, the following message is displayed:

  ```
  (FOC1071) VALUE FOR JOIN 'TO' FIELD OUT OF SEQUENCE. RETRIEVAL ENDED
  ```

- Many-to-many joins are not supported when the cross-referenced file is sequential. If multiple host field values match cross-referenced field values, the following message is generated:

  ```
  (FOC1072) DUPLICATES IN JOIN 'FROM' FIELD  : filename/value
  ```

- A comma-delimited file used as the cross-referenced file must consist of a single segment.

Example: Joining Two Comma-Delimited Files

The following request creates a comma-delimited file from the ITEMS data source and another comma-delimited file from the VIDEOTRK data source, sorted in the same order. It then joins the two comma-delimited files and issues a report request against the joined files. In order to issue a request against a FORMAT COM data source, the PCOMMA parameter is set ON:

```
TABLE FILE ITEMS
PRINT PRODNAME OURCOST RETAILPR ON_HAND
BY PRODCODE
ON TABLE HOLD AS ITEMCOM FORMAT COM
END

TABLE FILE VIDEOTRK
PRINT LASTNAME FIRSTNAME
BY PRODCODE
WHERE LASTNAME NE 'NON-MEMBER'
ON TABLE HOLD AS VIDCOM FORMAT COM
END

JOIN PRODCODE IN VIDCOM TO PRODCODE IN ITEMCOM AS J1

SET PCOMMA = ON

TABLE FILE VIDCOM
PRINT LASTNAME FIRSTNAME PRODNAME
WHERE PRODNAME LIKE '%SHIRT%'
END
```

The output is:

```
LASTNAME         FIRSTNAME   PRODNAME
--------         ---------   --------
WILLIAMS         KENNETH     T-SHIRT RED
WHITE            PATRICIA    T-SHIRT RED
CHANG            ROBERT      T-SHIRT RED
CRUZ             IVY         T-SHIRT BLUE
LEVINE           JOSHUA      T-SHIRT BLUE
CHANG            ROBERT      T-SHIRT BLUE
JOSEPH           JAMES       SWEATSHIRT
CHANG            ROBERT      SWEATSHIRT
```

Compiling DEFINEs With Missing Attributes

As of FOCUS 7.6.2, DEFINEs that have the missing attribute on either side of the equal sign can be compiled.

SUBFOOT WITHIN

How to:

Generate Subfoot Fields Calculated Within a Sort Break

Reference:

Usage Notes for SUBFOOT WITHIN

Example:

Generating a Subfoot Within a Sort Group

As of FOCUS 7.6.3, the WITHIN phrase can be added to a SUBFOOT command.

By adding the WITHIN phrase to a SUBFOOT command, you can create a subfooting whose fields are calculated within a sort field break. Without the WITHIN phrase, each SUBFOOT takes its field values from the report row directly above it on the report output. Calculating field values within a sort break is useful when applying a prefix operator to fields in the subfooting. If you do not specify a prefix operator when using WITHIN, the field is assigned the default prefix operator SUM. and produces a subtotal within the sort break.

Syntax: How to Generate Subfoot Fields Calculated Within a Sort Break

You can specify a subfoot in a BY phrase or in an ON phrase.

```
{ON|BY} byfield SUBFOOT [WITHIN] [MULTILINES]
         "text [<prefop.fieldname ... ]"
        [WHEN expression;]
```

where:

byfield

Is the sort field whose change in value generates the SUBFOOT.

WITHIN

Causes the fields in the SUBFOOT to be calculated within each value of *byfield*. Without this option, a field in the SUBFOOT is taken from the last line of report output above the subfooting.

text

Is the text you supply between double quotation marks that will be printed following the SUBFOOT phrase. The total number of subheads and subfoots in one report request may not exceed nine.

prefop

Is one of the following prefix operators: AVE., MIN., MAX., CNT., SUM., ASQ. If no prefix operator is specified, the default operator applied is SUM.

fieldname

Is field whose value will be printed in the subfooting calculated within the sort field value.

MULTILINES

Is used to suppress the SUBFOOT when there is only one line of output for the BY group. Note that MULTI-LINES is a synonym for MULTILINES.

expression

Specifies a conditional subfoot whose appearance on the report output is determined by a Boolean expression. WHEN must be placed on the line following the text you enclose in double quotation marks.

Reference: Usage Notes for SUBFOOT WITHIN

- ❑ SUBFOOT WITHIN is useful where a prefixed field within a sort break would result in a single value (for example, AVE., MIN, MAX). Use of PCT. or APCT. displays only the last value from the sort group.
- ❑ SUBFOOT WITHIN "<prefix.fieldname " does *not* result in the same value as SUBTOTAL prefix. The SUBFOOT WITHIN creates a display field that operates on the original input records. SUBTOTAL with a prefix operates on the internal matrix (so AVE. is the average of the SUMS or, if a display field had the prefix AVE., the average of the averages). SUBFOOT WITHIN "<AVE.field " generates an overall average.
- ❑ Prefix operators are not supported on alphanumeric fields in a WITHIN phrase.
- ❑ ST. is not supported in a SUBFOOT WITHIN phrase.

Example: Generating a Subfoot Within a Sort Group

The following request displays the average and minimum salary values first within department, then within department and job class, and last within department, job calss and employee ID. Subfootings are generated on the department and jobcode sort fields. The DEFINE FILE command created two additional fields with the SALARY value, one for each sort break:

```
DEFINE FILE EMPDATA
SALDEPT/D6  WITH SALARY = SALARY;
SALDEPTJOB/D6  WITH SALARY = SALARY;
DEPT/A4 WITH SALARY = EDIT(DEPT, '9999');
JOB/A8 WITH SALARY = JOBCLASS;
END
TABLE FILE EMPDATA
SUM AVE.SALDEPT    AS 'DEPT,AVE'
     MIN.SALDEPT   AS 'DEPT,MIN'
  BY DEPT
SUM AVE.SALDEPTJOB  AS 'DEPT/JOB,  AVE' IN 32
     MIN.SALDEPTJOB AS 'DEPT/JOB,  MIN' IN 42
BY DEPT
BY JOB
PRINT AVE.SALARY/D6  AS 'AVE' IN 52
      MIN.SALARY/D6  AS 'MIN' IN 61
BY DEPT
BY JOB
BY PIN    NOPRINT
ON DEPT SUBFOOT
"******************DEPARTMENT <DEPT SUBFOOT**************************"
"NOT WITHIN: AVE=<AVE.SALARY  MIN=<MIN.SALARY "
ON DEPT SUBFOOT WITHIN
"    WITHIN: AVE=<AVE.SALARY  MIN=<MIN.SALARY "
"*******************************************************************"
ON JOB SUBFOOT
"</1 ***************DEPARTMENT <DEPT / JOB <JOB  SUBFOOT***************"
"NOT WITHIN: AVE=<AVE.SALARY  MIN=<MIN.SALARY "
ON JOB SUBFOOT WITHIN
"    WITHIN: AVE=<AVE.SALARY  MIN=<MIN.SALARY "
"******************************************************************* </1"
  WHERE DEPT EQ 'MARK'
  WHERE  JOBCLASS EQ '257PSB' OR '257PTB'
END
```

The report output shows that each SUBFOOT without the WITHIN phrase uses the report line above the subfooting in the calculations. The SUBFOOT within both department and jobcode uses the calculations that were specified in the second SUM command (by department by jobcode), and the SUBFOOT within department only uses the calculations that were specified in the first SUM command (by department):

```
         DEPT     DEPT          DEPT/JOB  DEPT/JOB
DEPT     AVE      MIN   JOB       AVE       MIN         AVE       MIN
----     ----     ----  ---     --------  --------      ---       ---
MARK   56,757   50,500  257PSB   55,860    50,500    55,500    55,500
                                                     62,500    62,500
                                                     50,500    50,500
                                                     52,000    52,000
                                                     58,800    58,800

***************DEPARTMENT MARK / JOB 257PSB  SUBFOOT***************
NOT WITHIN: AVE=     $58,800.00  MIN=     $58,800.00
    WITHIN: AVE=     $55,860.00  MIN=     $50,500.00
*******************************************************************

                        257PTB   59,000    55,500    62,500    62,500
                                                     55,500    55,500

***************DEPARTMENT MARK / JOB 257PTB  SUBFOOT***************
NOT WITHIN: AVE=     $55,500.00  MIN=     $55,500.00
    WITHIN: AVE=     $59,000.00  MIN=     $55,500.00
*******************************************************************
******************DEPARTMENT MARK SUBFOOT**************************
NOT WITHIN: AVE=     $55,500.00  MIN=     $55,500.00
    WITHIN: AVE=     $56,757.14  MIN=     $50,500.00
*******************************************************************
```

3 Output Format Enhancements

This chapter describes enhancements in saving or holding report output.

For additional information, see *Increased Length for HOLD Files in FOCUS Format* in Chapter 8, *Raised Limits*.

Topics:

Saving Report Output in PowerPoint Format

How to:

Save Report Output in PowerPoint Format

Example:

Saving Report Output in PowerPoint Format

Report output can be styled and saved as a PowerPoint presentation. Each report page becomes a separate slide in the PowerPoint file.

Syntax: How to Save Report Output in PowerPoint Format

```
ON TABLE {HOLD|SAVE} [AS name] FORMAT PPT
```

where:

name

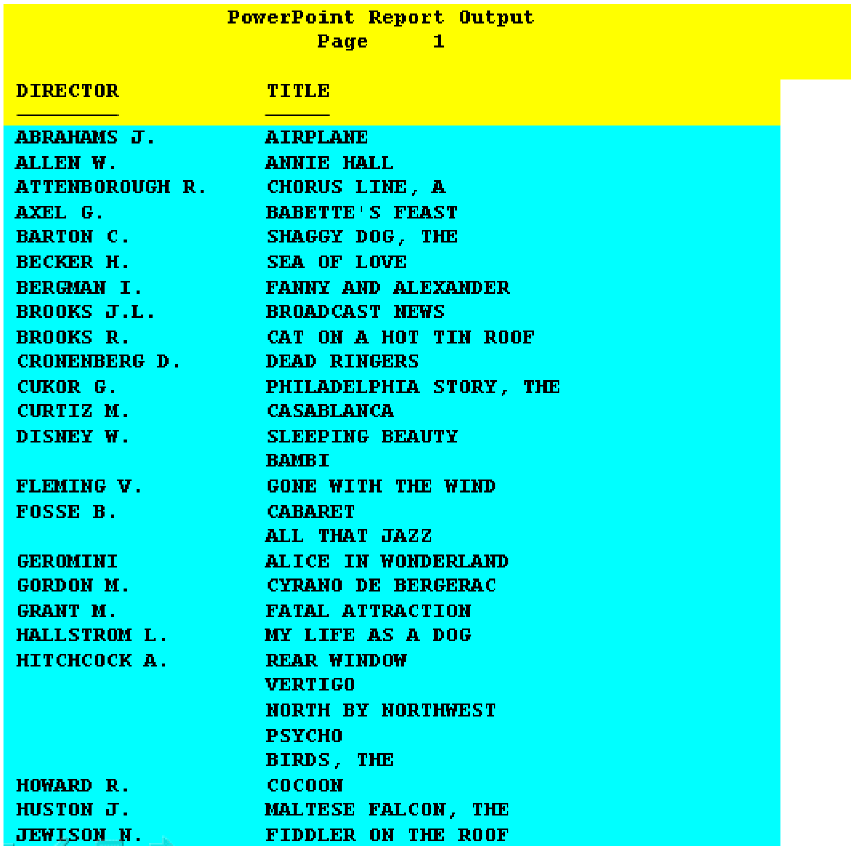
Assigns a name to the PowerPoint file. The file extension is ppt. The default name is HOLD.

Example: Saving Report Output in PowerPoint Format

The following request creates a two-page report using the MOVIES data source:

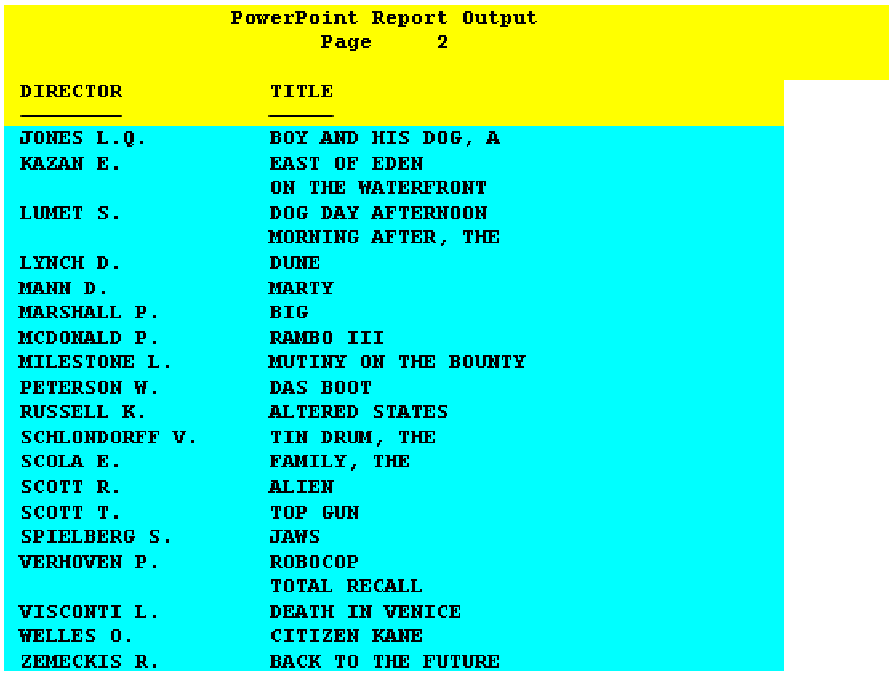
```
TABLE FILE MOVIES
HEADING CENTER
"PowerPoint Report Output"
"Page <TABPAGENO"
" "
PRINT TITLE
BY DIRECTOR
WHERE DIRECTOR NE ' '
ON TABLE SAVE AS MOVIE2 FORMAT PPT

ON TABLE SET STYLE *
type=report, style=bold, color=black, backcolor=yellow, $
type=tabheading, style=bold, color=black, colspan=2,
 backcolor=yellow, $
type=data, backcolor=aqua, $
ENDSTYLE
END
```

The output consists of two slides in a presentation named movie2.ppt. The first report page becomes the first slide.



```
                 PowerPoint Report Output
                        Page     1

DIRECTOR            TITLE
--------            -----
ABRAHAMS J.         AIRPLANE
ALLEN W.            ANNIE HALL
ATTENBOROUGH R.     CHORUS LINE, A
AXEL G.             BABETTE'S FEAST
BARTON C.           SHAGGY DOG, THE
BECKER H.           SEA OF LOVE
BERGMAN I.          FANNY AND ALEXANDER
BROOKS J.L.         BROADCAST NEWS
BROOKS R.           CAT ON A HOT TIN ROOF
CRONENBERG D.       DEAD RINGERS
CUKOR G.            PHILADELPHIA STORY, THE
CURTIZ M.           CASABLANCA
DISNEY W.           SLEEPING BEAUTY
                    BAMBI
FLEMING V.          GONE WITH THE WIND
FOSSE B.            CABARET
                    ALL THAT JAZZ
GEROMINI            ALICE IN WONDERLAND
GORDON M.           CYRANO DE BERGERAC
GRANT M.            FATAL ATTRACTION
HALLSTROM L.        MY LIFE AS A DOG
HITCHCOCK A.        REAR WINDOW
                    VERTIGO
                    NORTH BY NORTHWEST
                    PSYCHO
                    BIRDS, THE
HOWARD R.           COCOON
HUSTON J.           MALTESE FALCON, THE
JEWISON N.          FIDDLER ON THE ROOF
```

The second report page becomes the second slide.



```
                 PowerPoint Report Output
                        Page     2

DIRECTOR            TITLE
--------            -----
JONES L.Q.          BOY AND HIS DOG, A
KAZAN E.            EAST OF EDEN
                    ON THE WATERFRONT
LUMET S.            DOG DAY AFTERNOON
                    MORNING AFTER, THE
LYNCH D.            DUNE
MANN D.             MARTY
MARSHALL P.         BIG
MCDONALD P.         RAMBO III
MILESTONE L.        MUTINY ON THE BOUNTY
PETERSON W.         DAS BOOT
RUSSELL K.          ALTERED STATES
SCHLONDORFF V.      TIN DRUM, THE
SCOLA E.            FAMILY, THE
SCOTT R.            ALIEN
SCOTT T.            TOP GUN
SPIELBERG S.        JAWS
VERHOVEN P.         ROBOCOP
                    TOTAL RECALL
VISCONTI L.         DEATH IN VENICE
WELLES O.           CITIZEN KANE
ZEMECKIS R.         BACK TO THE FUTURE
```

Preserving Leading and Internal Blanks in Report Output

How to:

Preserve Leading and Internal Blanks in HTML and EXL2K Reports

Example:

Preserving Leading and Internal Blanks in HTML and EXL2K Report Output

By default, HTML browsers and Excel remove leading and trailing blanks from text and compress multiple internal blanks to a single blank.

If you want to preserve leading and internal blanks in HTML and EXL2K report output, you can issue the SET SHOWBLANKS=ON command.

Even if you issue this command, trailing blanks will not be preserved except in heading, subheading, footing, and subfooting lines that use the default heading or footing alignment.

Syntax: How to Preserve Leading and Internal Blanks in HTML and EXL2K Reports

In a FOCEXEC, on the command line, or in a profile use the following syntax

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

In a request, use the following syntax

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

where:

`OFF`

Removes leading blanks and compresses internal blanks in HTML and EXL2K report output.

`ON`

Preserves leading and internal blanks in HTML and EXL2K report output. Also preserves trailing blanks in heading, subheading, footing, and subfooting lines that use the default heading or footing alignment.

Example: Preserving Leading and Internal Blanks in HTML and EXL2K Report Output

The following request creates a virtual field that adds leading blanks to the value ACTION and both leading and internal blanks to the values TRAIN/EX and SCI/FI in the CATEGORY field. It also adds trailing blanks to the value COMEDY:

```
SET SHOWBLANKS = OFF
DEFINE FILE MOVIES
NEWCAT/A30 = IF CATEGORY EQ 'ACTION' THEN '  ACTION'
        ELSE IF CATEGORY EQ 'SCI/FI' THEN 'SCIENCE   FICTION'
        ELSE IF CATEGORY EQ 'TRAIN/EX' THEN '   TRANING    EXERCISE'
        ELSE IF CATEGORY EQ 'COMEDY' THEN 'COMEDY      '
        ELSE                'GENERAL';
END
TABLE FILE MOVIES
SUM CATEGORY LISTPR/D12.2 COPIES
BY NEWCAT
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
GRID=OFF,$
TYPE=REPORT, FONT=COURIER NEW,$
ENDSTYLE
END
```

With SHOWBLANKS OFF, these additional blanks are removed:

```
NEWCAT          CATEGORY   LISTPR COPIES
TRANING EXERCISETRAIN/EX   119.87     10
ACTION          ACTION      94.82     14
COMEDY          COMEDY     154.80     19
GENERAL         MYSTERY  1,216.82     67
SCIENCE FICTION SCI/FI     114.84      7
```

With SHOWBLANKS ON, the additional leading and internal blanks are preserved. Note that trailing blanks are not preserved:

```
NEWCAT                CATEGORY   LISTPR COPIES
   TRANING    EXERCISETRAIN/EX   119.87     10
  ACTION              ACTION      94.82     14
COMEDY                COMEDY     154.80     19
GENERAL               MYSTERY  1,216.82     67
SCIENCE   FICTION     SCI/FI     114.84      7
```

Structured HOLD Files

How to:

Activate Structured HOLD Files for a Request

Create a Structured HOLD File

Specify Options for Generating Structured HOLD Files

Reference:

Elements Included in a Structured HOLD File

Elements Not Included in a Structured HOLD File

Structural and Behavioral Notes

Example:

Creating a Structured HOLD File in ALPHA Format

Creating a Structured HOLD File in FOCUS Format

Reconstituting a Structured HOLD File

Excluding Fields From Structured HOLD Files

Structured HOLD Files facilitate migration of data sources and reports between operating environments.

Other HOLD formats capture data from the original sources and may retain some implicit structural elements from the request itself. However, they do not propagate most of the information about the original data sources accessed and their inter-relationships to the HOLD Master File or data source. Structured HOLD files, however, extract the data to a structure that parallels the original data sources. Subsequent requests against the HOLD file can use these retained data relationships to recreate the same types of relationships in other environments or in other types of data sources.

A Structured HOLD File can be created in ALPHA, BINARY, or FOCUS format:

- ❏ A Structured HOLD file created in either ALPHA or BINARY format is a flat file that saves the segment instances that contain the data that satisfy the conditions of the TABLE request. Multiple segments are generated based on the original structure read by the TABLE request. Segments are identified by assigning a RECTYPE for differentiation. Child segments in the original data source become a unique segment in the HOLD file
- ❏ A Structured HOLD file in FOCUS format uses normal FOCUS segments to retain the original structure.

In all cases the HOLD file contains all of the original segment instances required to provide the complete report based on the TABLE request itself. Regardless of the display command used in the original request (PRINT, LIST, SUM, COUNT), the Structured HOLD File is created as if the request used PRINT. Aggregation is ignored.

The HOLD file contains either all of the fields in the structure identified by the request that are used to satisfy the request, or all of the display fields and BY fields. The file does not contain DEFINE fields not specifically referenced in the request. It does contain all fields needed to evaluate any DEFINE fields referenced in the request.

Structured HOLD files are only supported for TABLE and TABLEF commands. They can be created anywhere a HOLD file is supported. You must activate Structured HOLD files in a specific request by issuing the ON TABLE SET EXTRACT command in the request prior to creating the Structured HOLD File.

Syntax: How to Activate Structured HOLD Files for a Request

```
ON TABLE SET EXTRACT {ON|*|OFF}
```

where:

`ON`

Activates Structured HOLD Files for this request and extracts all fields mentioned in the request.

`*`

Activates Structured HOLD Files for this request and indicates that a block of extract options follows. For example, you can exclude specific fields from the Structured HOLD File.

`OFF`

Deactivates Structured HOLD files for this request. OFF is the default value.

Syntax: How to Create a Structured HOLD File

Before issuing the HOLD command, activate Structured HOLD Files for the request by issuing the ON TABLE SET EXTRACT command described in *How to Activate Structured HOLD Files for a Request* on page 118. Then issue the HOLD command to create the Structured HOLD File:

```
[ON TABLE] {HOLD|PCHOLD} [AS name] FORMAT {ALPHA|BINARY|FOCUS}
```

where:

`name`

Is the name of the HOLD file. If omitted, the name becomes HOLD by default.

`FORMAT`

Is ALPHA, BINARY or FOCUS

Note: You can issue a SET command to set the default HOLD format to either ALPHA or BINARY:

```
SET HOLDFORMAT=ALPHA
SET HOLDFORMAT=BINARY
```

Syntax: How to Specify Options for Generating Structured HOLD Files

To specify options for creating the extract, such excluding specific fields, use the * option of the SET EXTRACT command:

```
ON TABLE SET EXTRACT *
EXCLUDE = (fieldname1, fieldname2, fieldname3, ..., fieldnamen),$
FIELDS={ALL|EXPLICIT},$
ENDEXTRACT
ON TABLE HOLD AS name FORMAT {ALPHA|BINARY|FOCUS}
```

where:

`EXCLUDE = (fieldname1, fieldname2, fieldname3, ..., fieldnamen)`

Excludes the specified fields from the HOLD file.

`,$`

Is required syntax for delimiting elements in the extract block.

`ALL`

Includes all real fields and all DEFINE fields that are used in running the request.

`EXPLICIT`

Includes only those real fields and DEFINE fields that are in the display list or the BY sort field listing. DEFINE fields that are not explicitly referenced, and fields that are used to evaluate DEFINEs, are not included.

```
ENDEXTRACT
```

Ends the extract block.

Example: Creating a Structured HOLD File in ALPHA Format

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
ON TABLE SET EXTRACT ON
ON TABLE HOLD FORMAT ALPHA
END
```

This request produces the following HOLD Master File:

```
FILENAME=HOLD    , SUFFIX=FIX     , $
  SEGMENT=EMPINFO, SEGTYPE=S0, $
    FIELDNAME=RECTYPE, ALIAS=R, USAGE=A3, ACTUAL=A3, $
    FIELDNAME=LAST_NAME, ALIAS='LN', USAGE=A15, ACTUAL=A15, $
    FIELDNAME=FIRST_NAME, ALIAS='FN', USAGE=A10, ACTUAL=A10, $
    FIELDNAME=DEPARTMENT, ALIAS='DPT', USAGE=A10, ACTUAL=A10, $
    FIELDNAME=ED_HRS, ALIAS='OJT', USAGE=F6.2, ACTUAL=A06, $
  SEGMENT=PAYINFO, SEGTYPE=S0, PARENT=EMPINFO, $
    FIELDNAME=RECTYPE, ALIAS=1, USAGE=A3, ACTUAL=A3, $
    FIELDNAME=SALARY, ALIAS='SAL', USAGE=D12.2M, ACTUAL=A12, $
    FIELDNAME=JOBCODE, ALIAS='JBC', USAGE=A3, ACTUAL=A03, $
```

Note the RECTYPE field generated for ALPHA or BINARY Structured HOLD files. Each record in the HOLD file begins with the RECTYPE to indicate the segment to which it belonged in the original structure. The root segment has RECTYPE=R. The RECTYPEs for other segments are sequential numbers assigned in top to bottom, left to right order.

Following are the first several records in the HOLD file:

```
R  STEVENS        ALFRED    PRODUCTION 25.00
1      11000.00A07
1      10000.00A07
R  SMITH          MARY      MIS        36.00
1      13200.00B14
R  JONES          DIANE     MIS        50.00
1      18480.00B03
1      17750.00B02
R  SMITH          RICHARD   PRODUCTION 10.00
1       9500.00A01
1       9050.00B01
```

Example: Creating a Structured HOLD File in FOCUS Format

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
ON TABLE SET EXTRACT ON
ON TABLE HOLD FORMAT FOCUS
END
```

This request produces the following HOLD Master File:

```
FILENAME=HOLD    , SUFFIX=FOC     , $
  SEGMENT=EMPINFO, SEGTYPE=S0, $
    FIELDNAME=LAST_NAME, ALIAS='LN', USAGE=A15, $
    FIELDNAME=FIRST_NAME, ALIAS='FN', USAGE=A10, $
    FIELDNAME=DEPARTMENT, ALIAS='DPT', USAGE=A10, $
    FIELDNAME=ED_HRS, ALIAS='OJT', USAGE=F6.2, $
  SEGMENT=PAYINFO, SEGTYPE=S0, PARENT=EMPINFO, $
    FIELDNAME=SALARY, ALIAS='SAL', USAGE=D12.2M, $
    FIELDNAME=JOBCODE, ALIAS='JBC', USAGE=A3, $
```

Example: Reconstituting a Structured HOLD File

The following request reconstitutes the original FOCUS data source from the Structured HOLD File created in *How to Creating a Structured HOLD File in ALPHA Format*:

```
TABLE FILE HOLD
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
ON TABLE SET EXTRACT ON
ON TABLE HOLD AS RECONST FORMAT FOCUS
END
```

This request produces the following Master File:

```
FILENAME=RECONST    , SUFFIX=FOC     , $
  SEGMENT=EMPINFO, SEGTYPE=S0, $
    FIELDNAME=LAST_NAME, ALIAS='LN', USAGE=A15, $
    FIELDNAME=FIRST_NAME, ALIAS='FN', USAGE=A10, $
    FIELDNAME=DEPARTMENT, ALIAS='DPT', USAGE=A10,
    FIELDNAME=ED_HRS, ALIAS='OJT', USAGE=F6.2, $
  SEGMENT=PAYINFO, SEGTYPE=S0, PARENT=EMPINFO, $
    FIELDNAME=SALARY, ALIAS='SAL', USAGE=D12.2M, $
    FIELDNAME=JOBCODE, ALIAS='JBC', USAGE=A3, $
```

The following request prints the report output:

```
TABLE FILE RECONST
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
END
```

The output is:

```
DEPARTMENT              SALARY  LAST_NAME          FIRST_NAME  JOBCODE  ED_HRS
----------              ------  ---------          ----------  -------  ------
MIS                 $27,062.00  CROSS              BARBARA     A17       45.00
                    $25,775.00  CROSS              BARBARA     A16       45.00
                    $21,780.00  BLACKWOOD          ROSEMARIE   B04       75.00
                    $18,480.00  JONES              DIANE       B03       50.00
                                MCCOY              JOHN        B02         .00
                    $17,750.00  JONES              DIANE       B02       50.00
                    $13,200.00  SMITH              MARY        B14       36.00
                     $9,000.00  GREENSPAN          MARY        A07       25.00
                     $8,650.00  GREENSPAN          MARY        B01       25.00
PRODUCTION          $29,700.00  BANNING            JOHN        A17         .00
                    $26,862.00  IRVING             JOAN        A15       30.00
                    $24,420.00  IRVING             JOAN        A14       30.00
                    $21,120.00  ROMANS             ANTHONY     B04        5.00
                    $16,100.00  MCKNIGHT           ROGER       B02       50.00
                    $15,000.00  MCKNIGHT           ROGER       B02       50.00
                    $11,000.00  STEVENS            ALFRED      A07       25.00
                    $10,000.00  STEVENS            ALFRED      A07       25.00
                     $9,500.00  SMITH              RICHARD     A01       10.00
                     $9,050.00  SMITH              RICHARD     B01       10.00
```

Example: Excluding Fields From Structured HOLD Files

This request excludes the SALARY field used for sequencing.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME JOBCODE ED_HRS
BY DEPARTMENT
BY HIGHEST SALARY
ON TABLE SET EXTRACT *
EXCLUDE=(SALARY),$
ENDEXTRACT
ON TABLE HOLD FORMAT FOCUS
END
```

This request produces the following HOLD Master File:

```
FILENAME=HOLD    , SUFFIX=FOC     , $
  SEGMENT=EMPINFO, SEGTYPE=S0, $
    FIELDNAME=LAST_NAME, ALIAS='LN', USAGE=A15, $
    FIELDNAME=FIRST_NAME, ALIAS='FN', USAGE=A10, $
    FIELDNAME=DEPARTMENT, ALIAS='DPT', USAGE=A10, $
    FIELDNAME=ED_HRS, ALIAS='OJT', USAGE=F6.2, $
  SEGMENT=PAYINFO, SEGTYPE=S0, PARENT=EMPINFO, $
    FIELDNAME=JOBCODE, ALIAS='JBC', USAGE=A3, $
```

Reference: Elements Included in a Structured HOLD File

Structured HOLD files contain all original segment instances required to complete the TABLE or TABLEF request. Regardless of the verb used in the original request (PRINT, LIST, SUM, or COUNT), the structured HOLD file will be created as if the verb was PRINT.

Specifically, the extract file contains the following elements:

❑ All real fields named in the request such as display objects, sort fields, and fields used in selection criteria (WHERE/IF tests).

  Note that fields referenced multiple times in a request are included only once in the HOLD file.

❑ Fields used in FILTER FILE condition.

❑ Prefix operators are ignored except for ALL. (which just affects the amount of data collected and does not imply a calculation).

❑ Field based reformatting (FIELD1/FIELD2=) causes the original field and the format field to be included.

- A GROUP field if referenced explicitly or when all of its members are referenced in the request.

  Note: If a group member is specifically excluded (EXCLUDE) or not referenced, its GROUP is not added to the extract Master File (this applies to nested and overlapping groups as well). If a GROUP and its elements are all named in a request, the GROUP is not added as a real field in the extract HOLD file.

- For FIELDS=ALL, all DEFINE fields used in the request become real fields in the structured HOLD File and are included along with other fields used in the DEFINE expression (including other DEFINE fields). Use EXCLUDE to reduce the number of fields included in the EXTRACT output.
- For FIELDS=EXPLICIT, display objects and sort fields are included. DEFINE fields become real fields if referenced in the request, but fields used to create them will not be included unless referenced explicitly. This reduces the number of fields returned in the request.

Reference: Elements Not Included in a Structured HOLD File

- Prefix operators on WHERE fields are evaluated in data selection but not included in the extract output.
- Prefix operators on display objects are ignored (except ALL).
- Using Structured HOLD File syntax in MATCH, MORE, and GRAPH requests produces error messages and exits the procedure.
- WHERE/IF TOTAL tests are not supported in Structured HOLD File requests and result in cancellation of the request.
- Reformatting of real fields is ignored (only the real field is included).
- Computed fields are not included, but fields used in COMPUTE expressions are included in the extract file.

Reference: Structural and Behavioral Notes

- Structured HOLD File requests are subject to the same limitations on number and size of fields as any other TABLE request.

Structural Notes

- The following SET parameters are turned off or ignored in Structured HOLD File requests:
    - AUTOINDEX
    - AUTOPATH
    - AUTOSTRATEGY
    - EXTHOLD
    - EXTSORT
    - HOLDATTR
    - All SET and ON TABLE SET commands used to control output format are ignored in creating the extract file.
- Alternate views are respected and reflected in the structure of the extract file.
- Indexed views specified in the request are respected and reflected in the structure of the output file.
- If a request would generate a file containing two independent orphan segments because the parent segment is specifically excluded, a dummy system segment is created in the to act as parent of the two unrelated segments. There is only one instance of data for that segment. Both orphan segments refer to that system segment as parent. If the parent is missing because it was not mentioned in the request, it is activated during the request and included as the parent the segments
- In the event that two unique (U) segments are included without the parent segment, the unique segments are converted to segments with SEGTYPE S0 that reference the system segment as parent.
- JOIN and JOIN WHERE structures are supported.

SQL Optimization Notes

- SQL optimization for aggregation must be turned off for EXTRACT requests.

BY/ACROSS/FOR Notes

- BY and ACROSS sort fields become additional display objects.
- BY. . .ROWS and ACROSS . . .COLUMNS function only as implicit WHEREs to limit field values included.
- FOR fields are included.
- RECAP fields are excluded (like COMPUTEs).

- Summarization fields referencing previously identified fields are ignored in creating Structured HOLD Files. These include: SUMMARIZE, RECOMPUTE, SUBTOTAL, SUB-TOTAL ACROSS-TOTAL, ROW-TOTAL and COLUMN-TOTAL.

Formatting Notes

- Structured HOLD File processing ignores all formatting elements, including: IN, OVER, NOPRINT, SUP-PRINT, FOLD-LINE, SKIP-LINE, UNDER-LINE, PAGE-BREAK,TABPAGENO, AS, and column title justification. However, fields referenced within formatting commands, such as HEADING, FOOTING, SUBHEAD, and SUBFOOT, and any WHEN expressions associated with them, are included.
- All STYLE and STYLESHEET commands are ignored in producing extract output.
- AnV and AnW fields are supported. TX fields are exported in FOCUS files only.
- In the event that the FIELDNAME and ALIAS are the same for a real field and that field is redefined as itself (possibly with a different format), two fields are created in the HOLD Master File with identical field names and aliases. In this situation, the second version of the field can never be accessed if referenced by name. You can use FIELDS=EXPLICIT to include only the second version of the field. The following DEFINE illustrates and example of creating a duplicate field name and alias:

  ```
  DEFINE FILE CAR
  COUNTRY/A25=COUNTRY;
  END
  ```

DBA Notes

- DBA controls on source files are respected when running Structured HOLD File requests with the exception of RESTRICT=NOPRINT, where fields named in a request are not displayed (such fields cannot be exported and should be specifically EXCLUDED).
- DBA restrictions do not carry over to the HOLD Master File.

Reconstituting Extract Files

- To reconstitute a FOCUS or flat file from a Structured HOLD file, you use the same syntax used to generate the Structured HOLD File. The ON TABLE SET EXTRACT syntax must be used to preserve multipath structures.
- All reconstituted FOCUS segments are SEGTYPE=S0, as neither KEY nor INDEX information is retained. An INDEX can be reinserted using REBUILD INDEX.

SAVB FORMAT INTERNAL

How to:

Create a SAVB File With FORMAT INTERNAL

The SAVB command now supports FORMAT INTERNAL, which prevents padding of alphanumeric fields and enables you to supply format overrides for integer and packed fields that should not be padded with zeros.

When you create a SAVB file, no Master File is created. In addition, you can supply your own DCB attributes, such as record format and record length, by issuing a TSO ALLOCATE, DYNAM ALLOCATE, or CMS FILEDEF command prior to creating the output file.

Syntax: How to Create a SAVB File With FORMAT INTERNAL

```
SET HOLDLIST = PRINTONLY
TABLE FILE filename
display_command fieldname/[In|Pn.d]..
ON TABLE SAVB AS name FORMAT INTERNAL
END
```

where:

`PRINTONLY`

Causes your report request to propagate the output file with only the specified fields displaying in the report output. If you do not issue this setting, an extra field containing the padded field length is included in the output file.

`fieldname/[In|Pn.d]`

Specifies formats for integer and packed fields for which you wish to suppress padding. These formats override the ACTUAL formats used for the display formats in the Master File.

Note that floating point double-precision (D) and floating pointing point single-precision (F) fields are not affected by SAVB FORMAT INTERNAL.

`FORMAT INTERNAL`

Saves the SAVB file without padding for specified integer and packed decimal fields.

Excel Named Ranges

How to:

Use Excel Named Ranges

Reference:

Rules for Excel Named Ranges

Support for Excel Named Ranges

Example:

Using Excel Named Ranges

An Excel Named Range is a name assigned to a specific group of cells within an Excel worksheet that can be easily referenced by FOCUS applications. The FOCUS StyleSheet language facilitates the generation of Named Ranges.

The use of Excel Named Ranges provides many benefits, including the following:

- ❑ Provides advantages over static cell references, including the ability of named range data areas to expand to include new data added during scheduled workbook updates.
- ❑ Enables easy setup of Excel worksheets, created by FOCUS applications, as an ODBC (Open Database Connectivity) data source.
- ❑ Provides accurate, consistent data feeds to advanced Excel worksheet applications, which eliminates manual activities that tend to result in errors.
- ❑ Simplifies the process of referencing data in multiple worksheets. This is especially useful when named ranges are added to the output of an Excel Template report.

Syntax: How to Use Excel Named Ranges

To create Excel Named Ranges, use

```
TYPE=type, IN-RANGES=rangename, $
```

where:

type

Identifies the FOCUS report component to be included in the range. Normally, both of the following are used together:

`DATA` adds the DATA element of the report to the named range (excludes heading, footing, and column titles).

`TITLE` adds the TITLE element of the report to the named range (includes all column titles).

Note: Multiple elements can be added to the same named range.

rangename

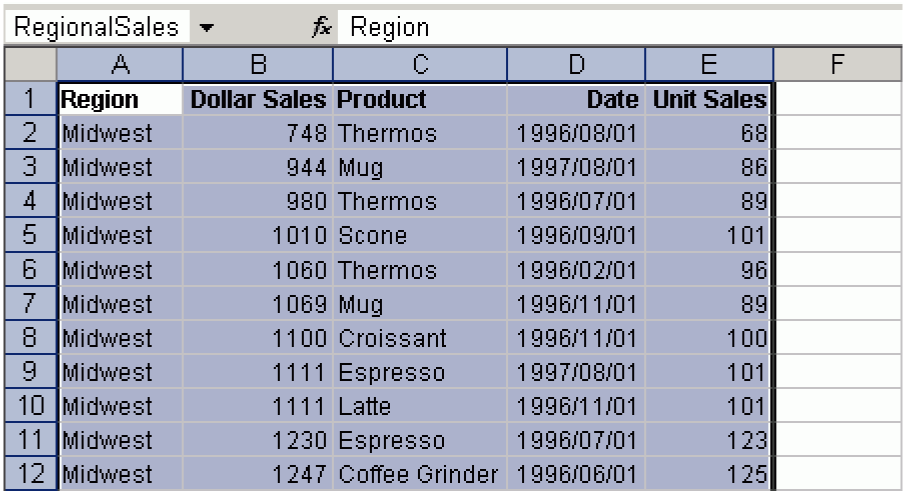
Is the name assigned to the output in the Excel workbook your application is creating, and is also the name that will be referenced by other FOCUS applications.

Example: Using Excel Named Ranges

This example creates one report in one worksheet of an Excel workbook. The code specific to Excel Named Ranges appears in bold in the following syntax.

```
TABLE FILE GGSALES
PRINT
     PRODUCT
     DATE
     UNITS
BY REGION
BY DOLLARS
ON TABLE SET PAGE-NUM OFF
ON TABLE SET BYDISPLAY ON
ON TABLE NOTOTAL
ON TABLE HOLD FORMAT EXL2K
ON TABLE SET STYLE *
     UNITS=IN, SQUEEZE=ON, ORIENTATION=PORTRAIT, $
TYPE=REPORT, FONT='ARIAL', SIZE=9, COLOR='BLACK', BACKCOLOR='NONE',
     STYLE=NORMAL, $
TYPE=DATA, IN-RANGES='RegionalSales', $
TYPE=TITLE, STYLE=BOLD, IN-RANGES='RegionalSales', $
ENDSTYLE
END
```

The Excel output is:



RegionalSales | Region

	A	B	C	D	E	F
1	**Region**	**Dollar Sales**	**Product**	**Date**	**Unit Sales**	
2	Midwest	748	Thermos	1996/08/01	68	
3	Midwest	944	Mug	1997/08/01	86	
4	Midwest	980	Thermos	1996/07/01	89	
5	Midwest	1010	Scone	1996/09/01	101	
6	Midwest	1060	Thermos	1996/02/01	96	
7	Midwest	1069	Mug	1996/11/01	89	
8	Midwest	1100	Croissant	1996/11/01	100	
9	Midwest	1111	Espresso	1997/08/01	101	
10	Midwest	1111	Latte	1996/11/01	101	
11	Midwest	1230	Espresso	1996/07/01	123	
12	Midwest	1247	Coffee Grinder	1996/06/01	125	

The name assigned to this Excel Named Range is RegionalSales. If additional rows of data are added, or columns of data are inserted, the named range will stretch to contain both new and existing data.

Reference: Rules for Excel Named Ranges

- The Excel data area associated with a named range must be continuous and cannot contain any breaks in the data. Examples of report components containing breaks in the data that cannot be part of a named range include SUBHEAD and SUBFOOT.
- It is recommended that you use ON TABLE SET BYDISPLAY ON. This activates the option to display repeated sort values, which produces continuous output with no breaks in the data.
- Two different worksheets from the same workbook cannot have the same range name.
- When creating Compound Excel reports (multiple TABLE requests output to the same Excel workbook), each report must have a unique range name that is stored at the workbook level.

Reference: Support for Excel Named Ranges

Excel Named Ranges are supported for the following Excel formats:

`EXL2K, EXL2K FORMULA, EXL2K TEMPLATE`

Excel Named Ranges are not supported for the following Excel formats:

`EXL2K BYTOC, EXCEL PIVOT`

Excel Named Ranges are not supported with any report syntax that produces discontinuous data or uses columnar references that span multiple columns, which includes the following:

`ACROSSCOLUMN, RECAP, RECOMPUTE, SUBHEAD, SUBFOOT, SUBTOTAL, SUB-TOTAL,`

Identifying Null Values in EXL2K Format Reports

How to:

Identify Null Values in EXL2K Report Output

Example:

Identifying Null Values in EXL2K Report Output

When a report is formatted as EXL2K, and null values are retrieved for one or more fields, blank spaces are displayed by default in each cell of the report output for the empty (null) fields. This behavior is the result of SET EMPTYCELLS ON being set by default in the background of all EXL2K reports. If you want to identify null values with something other than blank spaces, a character string can be used to populate all empty fields in a report.

Syntax: How to Identify Null Values in EXL2K Report Output

Outside of a report request, use the following syntax

```
SET NODATA = character_string
SET EMPTYCELLS = [ON|OFF]
```

In a report request, use the following syntax

```
ON TABLE SET NODATA character_string
ON TABLE SET EMPTYCELLS [ON|OFF]
```

where:

character_string

Is the string of characters displayed in the cells of the report for each field where null values are retrieved from the database. The maximum number of characters is 11. If the number of characters in the string exceeds the length of the output field, the additional characters will not be displayed. If special characters are used, the string must be enclosed in single quotes. SET EMPTYCELLS OFF must also be specified to make the SET NODATA command effective.

ON

Indicates that empty spaces are displayed in the cells of the report for each field where null values are retrieved from the database. ON is the default.

OFF

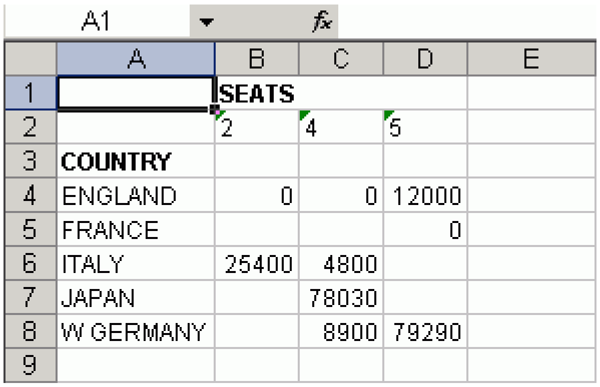
Indicates that zeros, or the character string specified with the SET NODATA command, will be displayed in the cells of the report for each field where null values are retrieved from the database. OFF must be specified when using SET NODATA.

Example: Identifying Null Values in EXL2K Report Output

The following syntax utilizes the default behavior of ON TABLE SET EMPTYCELLS ON, which is set in the background:

```
TABLE FILE CAR
   SUM SALES BY COUNTRY ACROSS SEATS
   ON TABLE HOLD FORMAT EXL2K
END
```

The following output displays empty spaces in the cells of the report for each field where null values are retrieved from the database:



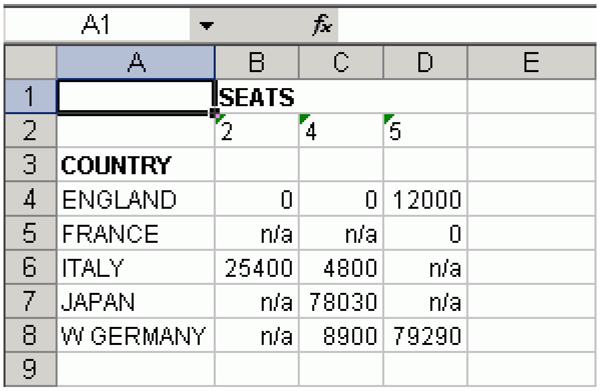
	A	B	C	D	E
1		**SEATS**			
2		2	4	5	
3	**COUNTRY**				
4	ENGLAND	0	0	12000	
5	FRANCE			0	
6	ITALY	25400	4800		
7	JAPAN		78030		
8	W GERMANY		8900	79290	
9					

The following syntax utilizes the SET NODATA command:

```
TABLE FILE CAR
   SUM SALES BY COUNTRY ACROSS SEATS
   ON TABLE SET NODATA 'n/a'
   ON TABLE SET EMPTYCELLS OFF
   ON TABLE HOLD FORMAT EXL2K
END
```

Note: If you do not add SET EMPTYCELLS OFF, the SET NODATA command will be ignored.

The following output displays 'n/a' in the cells of the report for each field where null values are retrieved from the database:

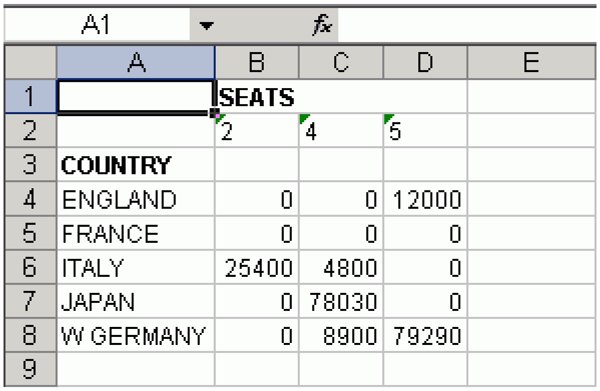


	A	B	C	D	E
1		SEATS			
2		2	4	5	
3	COUNTRY				
4	ENGLAND	0	0	12000	
5	FRANCE	n/a	n/a	0	
6	ITALY	25400	4800	n/a	
7	JAPAN	n/a	78030	n/a	
8	W GERMANY	n/a	8900	79290	
9					

The following syntax turns off the default SET EMPTYCELLS behavior and does not use SET NODATA, which makes it impossible to distinguish null values from zero quantities:

```
TABLE FILE CAR
   SUM SALES BY COUNTRY ACROSS SEATS
   ON TABLE SET EMPTYCELLS OFF
   ON TABLE HOLD FORMAT EXL2K
END
```

The following output displays zeros in the cells of the report for each field where either null values are retrieved from the database or the quantity is zero:



	A	B	C	D	E
1		**SEATS**			
2		2	4	5	
3	**COUNTRY**				
4	ENGLAND	0	0	12000	
5	FRANCE	0	0	0	
6	ITALY	25400	4800	0	
7	JAPAN	0	78030	0	
8	W GERMANY	0	8900	79290	
9					

Creating PDF Files for Use With UNIX Systems

How to:

Specify Line Termination Characters When Creating a PDF File

Reference:

Required PDFLINETERM Settings Based on Environment

PDF files created with HOLD FORMAT PDF present a challenge if you work in an MVS or VM environment and use UNIX-based systems as the server for Adobe or as an intermediate transfer point.

The end of each PDF file has a table containing the byte offset, including line termination characters, of each PDF object in the file. The offsets indicate that each line is terminated by two characters, a carriage return and a line feed, which is the standard Windows text file format. However, records in a UNIX text file are terminated by one character, a line feed only. When using default settings, the offsets in a PDF file will be incorrect, causing an error when Acrobat attempts to open the file. If the file is then transferred in BINARY mode to Windows, it cannot be opened in Acrobat for Windows, as the carriage-return character was not inserted.

One solution has been to transfer the file to the UNIX system in text mode and then transfer in text mode to the Windows system, as the carriage return is added by the transfer facility when transferring to Windows.

If that is not possible or desirable, you can use the SET PDFLINETERM=SPACE command to facilitate binary transfer to Windows from an ASCII-based UNIX system. This command causes an extra space character to be appended to each record of the PDF output file. This extra space acts as a placeholder for the expected carriage return character and makes the object offsets in the file correct when it is transferred from MVS or VM to a UNIX system. This enables a UNIX server to open a PDF file in that environment.

Note: A text mode transfer is always required when transferring a text file from a mainframe to any other environment (Windows, ASCII Unix, or EBCDIC Unix).

Syntax: How to Specify Line Termination Characters When Creating a PDF File

In a profile, a FOCEXEC, or from the command line, issue the following command:

```
SET PDFLINETERM={STANDARD|SPACE}
```

In a TABLE request, issue the following command:

```
ON TABLE SET PDFLINETERM {STANDARD|SPACE}
```

where:

`STANDARD`

Creates a PDF file without any extra characters. This file will be a valid PDF file if transferred in text mode to a Windows machine, but not to a UNIX machine. If subsequently transferred from a UNIX machine to a Windows machine in text mode, it will be a valid PDF file on the Windows machine.

`SPACE`

Creates a PDF file with an extra space character appended to each record. This file will be a valid PDF file if transferred in text mode to a UNIX machine, but not to a Windows machine. If subsequently transferred from an ASCII UNIX machine to a Windows machine in binary mode, it will be a valid PDF file on the Windows machine.

Reference: Required PDFLINETERM Settings Based on Environment

The following chart will assist you in determining the correct setting to use, based on your environment:

Transferring from MVS or VM to:	**SET PDFLINETERM=**
EBCDIC UNIX (text transfer)	SPACE
ASCII UNIX (text transfer)	SPACE
ASCII UNIX (text); then to Windows (binary)	SPACE
UNIX (text); then to Windows (text)	STANDARD
Directly to Windows (text)	STANDARD

Excel Table of Contents

Excel Table of Contents (TOC) is a feature that enables you to generate a multiple worksheet report in which a separate worksheet is generated for each value of the first BY field in the FOCUS report.

Note: This feature can be used only with Excel 2002 or higher releases because it requires the Web Archive file format, which was not available in Excel 2000 and earlier releases.

Syntax: How to Use the Excel Table of Contents Feature

Only a single BY field is allowed in an EXL2K Table of Contents report.

```
ON TABLE HOLD FORMAT EXL2K BYTOC
```

Since only one level of TOC is allowed for EXL2K reports, the optional number following the BYTOC keyword can only be 1.

The SET COMPOUND syntax, which precedes the TABLE command, may also be used to specify that a TOC be created:

```
SET COMPOUND=BYTOC
```

Since a TOC report is burst into worksheets according to the value of the first BY field in the report, the report must contain at least one BY field. The bursting field may be a NOPRINT field.

Reference: How to Name Worksheets

- The worksheet tab names are the BY field values that correspond to the data on the current worksheet. If the user specifies the TITLETEXT keyword in the stylesheet, it will be ignored.
- Excel limits the length of worksheet titles to 31 characters. The following special characters cannot be used: ':', '?', '*', and '/'.

Reference: Limitations of TOC Reports

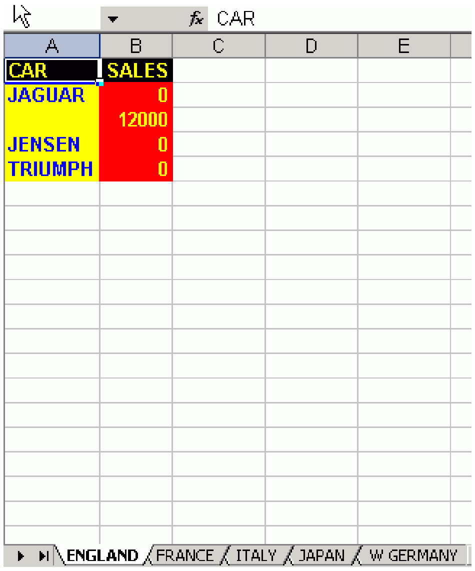
- A TOC report cannot be embedded in a compound report.
- A TOC report cannot be a pivot table report.

Note: FORMULA is not supported with bursting.

Example: Creating a Simple TOC Report

```
SET COMPOUND=BYTOC
TABLE FILE CAR
PRINT SALES BY COUNTRY NOPRINT BY CAR
ON TABLE HOLD FORMAT EXL2K
ON TABLE SET STYLE *
type=report, style=bold, color=yellow, backcolor=black, $
type=data, backcolor=red, $
type=data, column=car, color=blue, backcolor=yellow, $
END
```

The output is:



CAR	SALES
JAGUAR	0
	12000
JENSEN	0
TRIUMPH	0

Excel Compound Reports

Reference:

Guidelines for Using the OPEN, CLOSE, and NOBREAK Keywords and SET COMPOUND

Guidelines for Producing Excel Compound Reports

Example:

Creating a Simple Compound Report Using EXL2K

Creating a Compound Report With Pivot Tables and Formulas

Creating a Compound Report Using NOBREAK

Excel Compound Reports is a feature which enables you to generate multiple worksheet reports using the EXL2K output format.

The syntax of this feature is identical to that of PDF Compound Reports. By default, each of the component reports from the compound report is placed in a new Excel worksheet (analogous to a new page in PDF). If the NOBREAK keyword is used, the next report follows the current report on the same worksheet (analogous to starting the report on the same page in PDF).

Output is generated in Microsoft's Web Archive format. This format is labeled Single File Web Page in Excel's Save As dialog. Excel provides the conventionally given file types: MHT or MHTML. FOCUS uses the same XHT file type that is used for EXL2K reports.

The components of an Excel compound report can be FORMULA or PIVOT reports (subject to the restrictions). They cannot be Table of Contents (TOC) reports.

Note: Excel 2002 (Office XP) or higher must be installed. This feature will not work with earlier versions of Excel since they do not support the Web Archive file format.

Reference: Guidelines for Using the OPEN, CLOSE, and NOBREAK Keywords and SET COMPOUND

As with PDF, the keywords OPEN, CLOSE, and NOBREAK are used to control Excel compound reports. They can be specified with the HOLD command or with a separate SET COMPOUND command.

- OPEN is used on the first report of a sequence of component reports to specify that a compound report be produced.
- CLOSE is used to designate the last report in a compound report.
- NOBREAK specifies that the next report be placed on the same worksheet as the current report. If it is not present, the default behavior is to place the next report on a separate worksheet.

  NOBREAK may appear with OPEN on the first report, or alone on a report between the first and last reports. (Using CLOSE is irrelevant, since it refers to the placement of the next report, and no report follows the final report on which CLOSE appears.)
- When used with the HOLD syntax, the compound report keywords OPEN, CLOSE, and NOBREAK must appear immediately after FORMAT EXL2K, and before any additional keywords, such as FORMULA or PIVOT. For example, you can specify:
  - `ON TABLE HOLD FORMAT EXL2K OPEN`
  - `ON TABLE HOLD AS MYHOLD FORMAT EXL2K OPEN NOBREAK`
  - `ON TABLE HOLD FORMAT EXL2K NOBREAK FORMULA`
  - `ON TABLE HOLD FORMAT EXL2K CLOSE PIVOT PAGEFIELDS COUNTRY`
- As with PDF compound reports, compound report keywords can be alternatively specified using SET COMPOUND:
  - `SET COMPOUND = OPEN`
  - `SET COMPOUND = 'OPEN NOBREAK'`
  - `SET COMPOUND = NOBREAK`
  - `SET COMPOUND = CLOSE`

Reference: Guidelines for Producing Excel Compound Reports

- **Pivot Tables and NOBREAK.** Pivot Table Reports may appear in compound reports, but they may not be combined with another report on the same worksheet using NOBREAK.
- **Naming of Worksheets.** The default worksheet tab names will be Sheet1, Sheet2, and so on. You have the option to specify a different worksheet tab name by using the TITLETEXT keyword in the stylesheet. For example:

  ```
  TYPE=REPORT, TITLETEXT='Summary Report', $
  ```

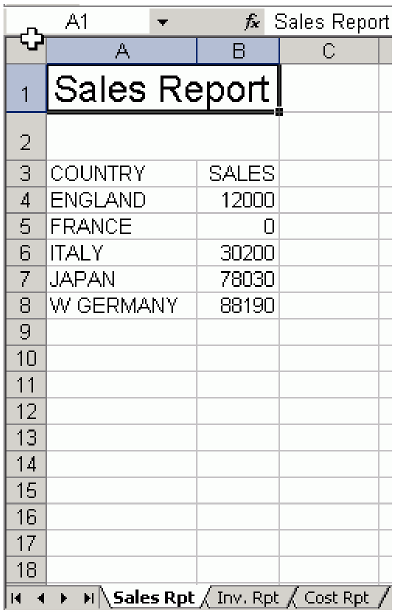
  Excel limits the length of worksheet titles to 31 characters. The following special characters cannot be used: ':', '?', '*', and '/'.
- **File Names and Formats.** The output file name (AS name, or HOLD by default) is obtained from the first report of the compound report (the report with the OPEN keyword). Output file names on subsequent reports are ignored.

  The HOLD FORMAT syntax used in the first component report in a compound report applies to all subsequent reports in the compound report, regardless of their format.
- **NOBREAK Behavior.** When NOBREAK is specified, the following report appears on the row immediately after the last row of the report with the NOBREAK. If additional spacing is required between the reports, a FOOTING or an ON TABLE SUBFOOT can be placed on the report with the NOBREAK, or a HEADING or an ON TABLE SUBHEAD can be placed on the following report. This allows the most flexibility, since if blank rows were added by default there would be no way to remove them.

Example: Creating a Simple Compound Report Using EXL2K

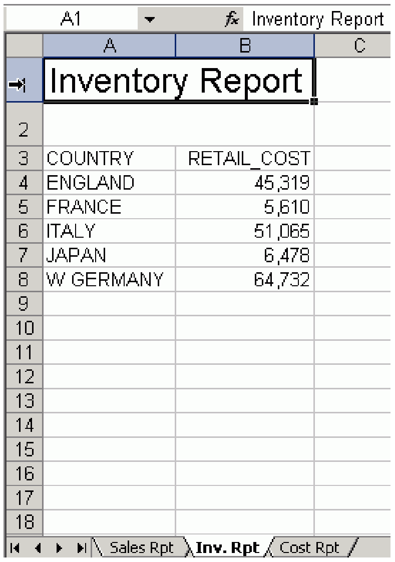
```
SET PAGE-NUM=OFF
TABLE FILE CAR
HEADING
"Sales Report"
" "
SUM SALES
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Sales Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD AS EX1 FORMAT EXL2K OPEN
END

TABLE FILE CAR
HEADING
"Inventory Report"
" "
SUM RC
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Inv. Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD FORMAT EXL2K
END

TABLE FILE CAR
HEADING
"Cost of Goods Sold Report"
" "
SUM DC
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Cost Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD FORMAT EXL2K CLOSE
END
```

The output for each tab in the Excel worksheet is:



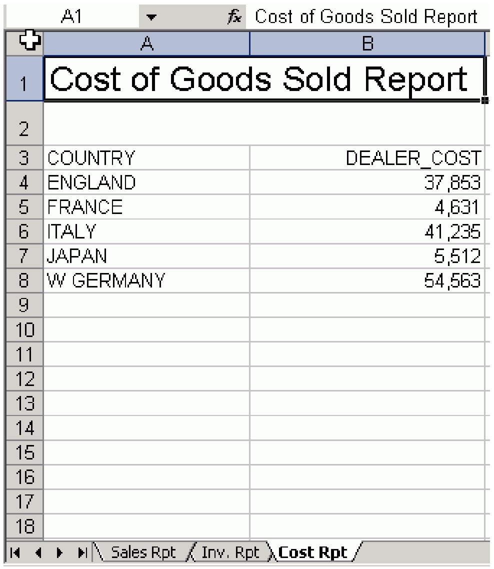
Sales Report

COUNTRY	SALES
ENGLAND	12000
FRANCE	0
ITALY	30200
JAPAN	78030
W GERMANY	88190



Inventory Report

COUNTRY	RETAIL_COST
ENGLAND	45,319
FRANCE	5,610
ITALY	51,065
JAPAN	6,478
W GERMANY	64,732



A1 Cost of Goods Sold Report

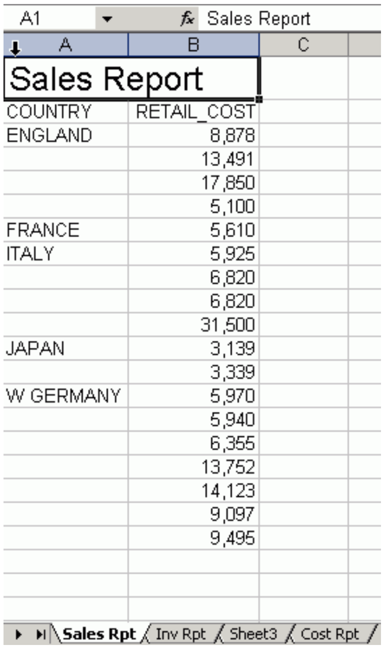
	A	B
1	Cost of Goods Sold Report	
2		
3	COUNTRY	DEALER_COST
4	ENGLAND	37,853
5	FRANCE	4,631
6	ITALY	41,235
7	JAPAN	5,512
8	W GERMANY	54,563
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		

Example: Creating a Compound Report With Pivot Tables and Formulas

```
SET PAGE-NUM=OFF
TABLE FILE CAR
HEADING
"Sales Report"
" "
PRINT RCOST
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Sales Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD AS PIV1 FORMAT EXL2K OPEN
END
```

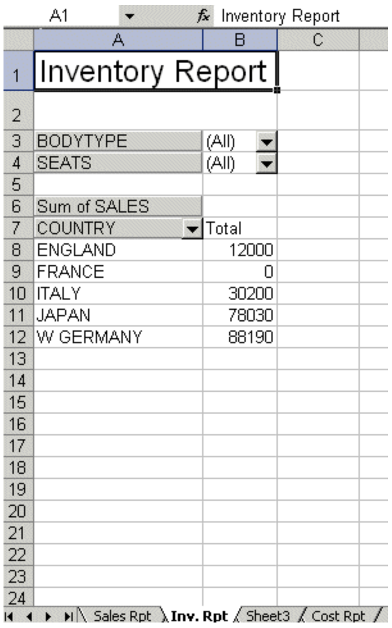
```
TABLE FILE CAR
HEADING
"Inventory Report"
" "
PRINT SALES
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Inv. Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD AS PPPP FORMAT EXL2K PIVOT
PAGEFIELDS TYPE SEATS
CACHEFIELDS MODEL MPG RPM
END

TABLE FILE CAR
SUM RCOST
BY COUNTRY BY CAR BY MODEL BY TYPE BY SEATS SUMMARIZE
ON MODEL SUB-TOTAL
ON TABLE HOLD AS XFOCB FORMAT EXL2K FORMULA
END

TABLE FILE CAR
HEADING
"Cost of Goods Sold Report"
" "
PRINT DCOST
BY COUNTRY
ON TABLE SET STYLE *
type=report, titletext='Cost Rpt', $
type=heading, size=18, $
ENDSTYLE
ON TABLE HOLD AS ONE FORMAT EXL2K CLOSE PIVOT
PAGEFIELDS RCOST
CACHEFIELDS MODEL TYPE SALES ACCEL SEATS
END
```

The output for each tab in the Excel worksheet is:



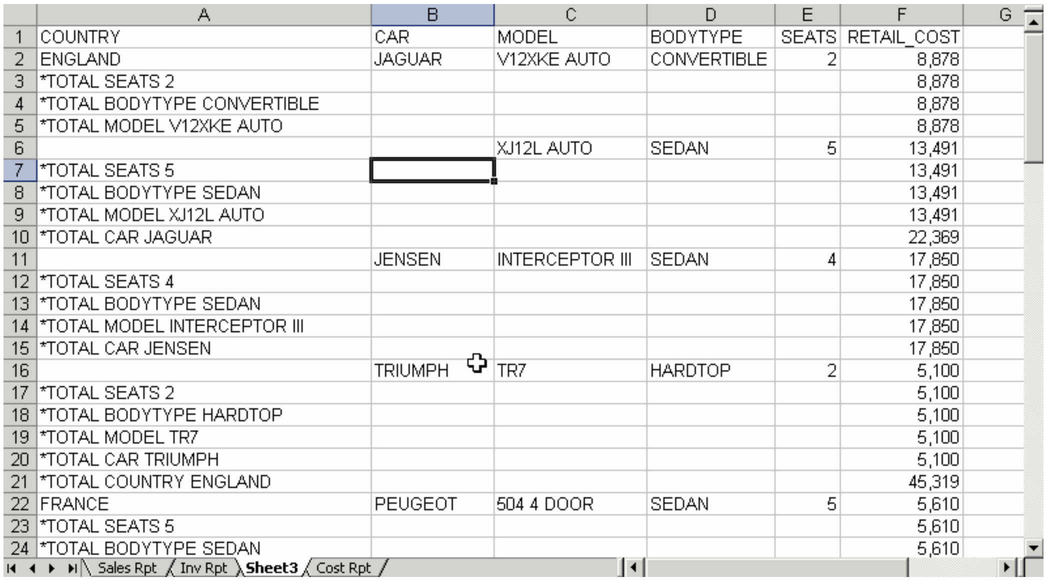
A1 Sales Report

A	B	C
Sales Report		
COUNTRY	RETAIL_COST	
ENGLAND	8,878	
	13,491	
	17,850	
	5,100	
FRANCE	5,610	
ITALY	5,925	
	6,820	
	6,820	
	31,500	
JAPAN	3,139	
	3,339	
W GERMANY	5,970	
	5,940	
	6,355	
	13,752	
	14,123	
	9,097	
	9,495	

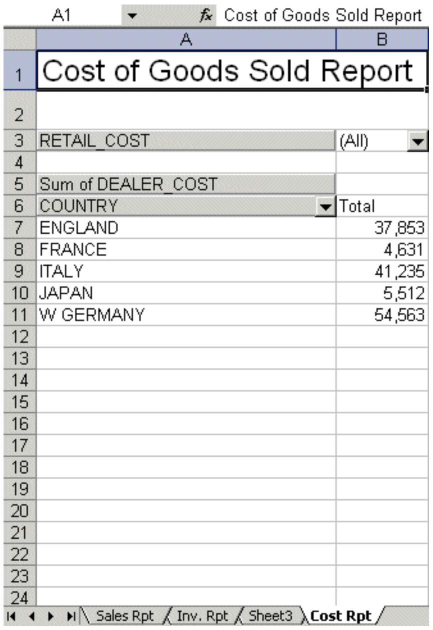


A1 Inventory Report

	A	B	C
1	Inventory Report		
2			
3	BODYTYPE	(All)	
4	SEATS	(All)	
5			
6	Sum of SALES		
7	COUNTRY	Total	
8	ENGLAND	12000	
9	FRANCE	0	
10	ITALY	30200	
11	JAPAN	78030	
12	W GERMANY	88190	
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			



	A	B	C	D	E	F	G
1	COUNTRY	CAR	MODEL	BODYTYPE	SEATS	RETAIL_COST	
2	ENGLAND	JAGUAR	V12XKE AUTO	CONVERTIBLE	2	8,878	
3	*TOTAL SEATS 2					8,878	
4	*TOTAL BODYTYPE CONVERTIBLE					8,878	
5	*TOTAL MODEL V12XKE AUTO					8,878	
6			XJ12L AUTO	SEDAN	5	13,491	
7	*TOTAL SEATS 5					13,491	
8	*TOTAL BODYTYPE SEDAN					13,491	
9	*TOTAL MODEL XJ12L AUTO					13,491	
10	*TOTAL CAR JAGUAR					22,369	
11		JENSEN	INTERCEPTOR III	SEDAN	4	17,850	
12	*TOTAL SEATS 4					17,850	
13	*TOTAL BODYTYPE SEDAN					17,850	
14	*TOTAL MODEL INTERCEPTOR III					17,850	
15	*TOTAL CAR JENSEN					17,850	
16		TRIUMPH	TR7	HARDTOP	2	5,100	
17	*TOTAL SEATS 2					5,100	
18	*TOTAL BODYTYPE HARDTOP					5,100	
19	*TOTAL MODEL TR7					5,100	
20	*TOTAL CAR TRIUMPH					5,100	
21	*TOTAL COUNTRY ENGLAND					45,319	
22	FRANCE	PEUGEOT	504 4 DOOR	SEDAN	5	5,610	
23	*TOTAL SEATS 5					5,610	
24	*TOTAL BODYTYPE SEDAN					5,610	



A1 Cost of Goods Sold Report

	A	B
1	Cost of Goods Sold Report	
2		
3	RETAIL_COST	(All)
4		
5	Sum of DEALER_COST	
6	COUNTRY	Total
7	ENGLAND	37,853
8	FRANCE	4,631
9	ITALY	41,235
10	JAPAN	5,512
11	W GERMANY	54,563
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		

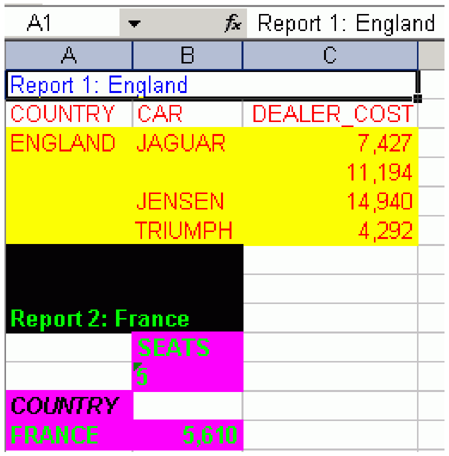
Example: Creating a Compound Report Using NOBREAK

In this example, the first two reports are on the first worksheet, and the last two reports are on the second worksheet, since NOBREAK appears on both the first and third reports.

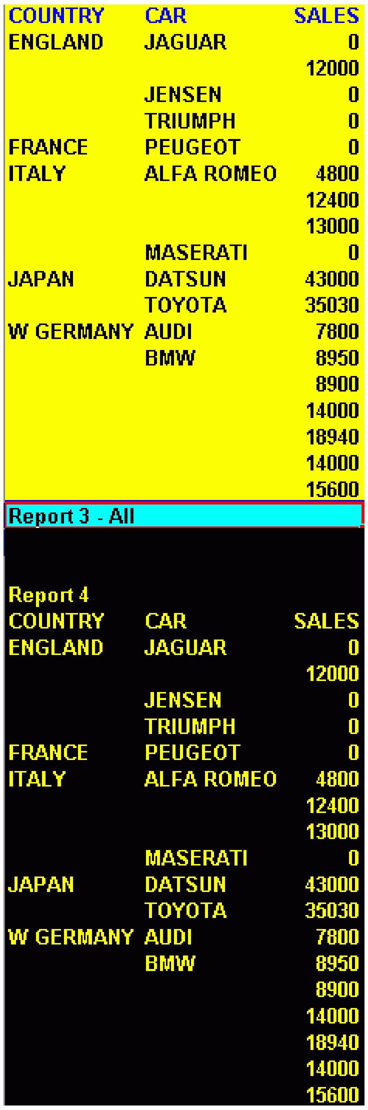
```
TABLE FILE CAR
HEADING
"Report 1: England"
PRINT DCOST BY COUNTRY BY CAR
IF COUNTRY EQ ENGLAND
ON TABLE HOLD FORMAT EXL2K OPEN NOBREAK
ON TABLE SET STYLE *
type=report, color=red, $
type=data, backcolor=yellow, $
type=heading, color=blue, $
END

TABLE FILE CAR
HEADING
" "
" "
"Report 2: France"
PRINT RCOST BY COUNTRY
IF COUNTRY EQ FRANCE
ACROSS SEATS
ON TABLE HOLD FORMAT EXL2K
ON TABLE SET STYLE *
type=report, color=lime, backcolor=fuschia, style=bold, $
type=title, color=black, style=bold+italic, $
type=heading, backcolor=black, $
END

TABLE FILE CAR
FOOTING
"Report 3 - All"
PRINT SALES BY COUNTRY BY CAR
ON TABLE HOLD FORMAT EXL2K NOBREAK
ON TABLE SET STYLE *
type=report, backcolor=yellow, style=bold, $
type=title, color=blue, $
type=footing, backcolor=aqua, $
END
```

```
TABLE FILE CAR
HEADING
" "
" "
"Report 4"
PRINT SALES BY COUNTRY BY CAR
ON TABLE HOLD FORMAT EXL2K CLOSE
ON TABLE SET STYLE *
type=report, color=yellow, backcolor=black, style=bold, $
END
```

The output is:



	A	B	C
	Report 1: England		
	COUNTRY	CAR	DEALER_COST
	ENGLAND	JAGUAR	7,427
			11,194
		JENSEN	14,940
		TRIUMPH	4,292
	Report 2: France		
		SEATS	
		5	
	COUNTRY		
	FRANCE	5,610	



COUNTRY	CAR	SALES
ENGLAND	JAGUAR	0
		12000
	JENSEN	0
	TRIUMPH	0
FRANCE	PEUGEOT	0
ITALY	ALFA ROMEO	4800
		12400
		13000
	MASERATI	0
JAPAN	DATSUN	43000
	TOYOTA	35030
W GERMANY	AUDI	7800
	BMW	8950
		8900
		14000
		18940
		14000
		15600

Report 3 - All

Report 4

COUNTRY	CAR	SALES
ENGLAND	JAGUAR	0
		12000
	JENSEN	0
	TRIUMPH	0
FRANCE	PEUGEOT	0
ITALY	ALFA ROMEO	4800
		12400
		13000
	MASERATI	0
JAPAN	DATSUN	43000
	TOYOTA	35030
W GERMANY	AUDI	7800
	BMW	8950
		8900
		14000
		18940
		14000
		15600

Displaying An and AnV Fields With Line Breaks

How to:

Display An and AnV Fields Containing Line Breaks on Multiple Lines

Example:

Displaying an Alphanumeric Field With Line Breaks in a PDF Report

Using an Alphanumeric Field With a LIne Break in a Subfoot

Using StyleSheet attributes, you can display An (character) and AnV (varchar) fields that contain line breaks on multiple lines in a PDF or PostScript report. Line breaks can be based on line feeds, carriage returns, or a combination of both. If you do not add these StyleSheet attributes, all line feed and carriage return formatting within these fields will be ignored.

Syntax: How to Display An and AnV Fields Containing Line Breaks on Multiple Lines

```
TYPE=REPORT,LINEBREAK='type',$
```

where:

`REPORT`

Is the required component for the LINEBREAK attribute.

`'type'`

Specifies that line breaks will be inserted in a report based on the following:

`LF` inserts a line break after each line-feed character found in all An and AnV fields.

`CR` inserts a line break after each carriage-return character found in all An and AnV fields.

`LFCR` inserts a line break after each combination of a line-feed character followed by a carriage-return character found in all An and AnV fields.

`CRLF` inserts a line break after each combination of a carriage-return character followed by a line-feed character found in all An and AnV fields.

Note: The report output must be formatted as PDF or PostScript.

Example: Displaying an Alphanumeric Field With Line Breaks in a PDF Report

The following request defines an alphanumeric named ANLB field with a semi-colon in the middle. The CTRAN function then replaces the semi-colon with a carriage return character and stores this string in a field named ANLBC. On the report output, this field displays on two lines:

```
DEFINE FILE EMPLOYEE
ANLB/A40 ='THIS IS AN An FIELD;WITH A LINE BREAK.';
ANLBC/A40 = CTRAN(40, ANLB, 094, 013  , ANLBC);
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME ANLBC
WHERE LAST_NAME EQ 'BLACKWOOD'
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT,LINEBREAK='CR',$
ENDSTYLE
END
```

The output is:

```
LAST_NAME           ANLBC

BLACKWOOD           THIS IS AN An FIELD
                    WITH A LINE BREAK.
```

Example: Using an Alphanumeric Field With a LIne Break in a Subfoot

The following request defines an alphanumeric named ANLB field with a semi-colon in the middle. The CTRAN function then replaces the semi-colon with a carriage return character and stores this string in a field named ANLBC. In the subfoot, this field displays on two lines:

```
DEFINE FILE EMPLOYEE
ANLB/A40 ='THIS IS AN An FIELD;WITH A LINE BREAK.';
ANLBC/A40 = CTRAN(40, ANLB, 094, 013   , ANLBC);
END
TABLE FILE EMPLOYEE
PRINT FIRST_NAME
BY LAST_NAME
WHERE LAST_NAME EQ 'BLACKWOOD'
ON LAST_NAME SUBFOOT
  " "
  " <ANLBC "
ON TABLE HOLD FORMAT PDF
ON TABLE SET STYLE *
TYPE=REPORT,LINEBREAK='CR',$
ENDSTYLE
END
```

The output is:

```
LAST_NAME         FIRSTNAME

BLACKWOOD         ROSEMARIE

  THIS  IS AN An FIELD
  WITH A LINE BREAK.
```

Creating HTML Reports With Absolute Positioning

How to:

Create Web Archive Report Output

Create a DHTML Report

Reference:

Usage Notes for Format DHTML

Example:

Creating a DHTML Report

Format DHTML provides HTML output that has most of the features normally associated with output formatted for printing such as PDF or PostScript output. You can create an HTML file (.htm) or a Web Archive file (.mht). The type of output file produced is controlled by the value of the HTMLARCHIVE parameter.

Some of the features supported by format DHTML are:

- **Absolute positioning.** DHTML precisely places text and images inside an HTML report, allowing you to use the same StyleSheet syntax to lay out HTML as you use for PDF or PS output.
- **On demand paging.** On demand paging is available with SET HTMLARCHIVE=OFF.
- **PDF StyleSheet features.** For example, the following features are supported: grids, background colors, OVER, .

Syntax: How to Create Web Archive Report Output

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

where:

`ON`

Creates output in Web Archive format. The file type of the output file is MHT.

`OFF`

Creates output in HTML format The file type of the output file is HTM. OFF is the default value.

Syntax: How to Create a DHTML Report

```
[ON TABLE] HOLD [AS name] FORMAT DHTML
```

where:

name

Specifies the name of the output file. The extension will be HTML if SET HTMLARCHIVE is OFF or MHTif SET HTMLARCHIVE is ON.

Reference: Usage Notes for Format DHTML

- ❑ The font map file for DHTML reports is DHTML FOCFTMAP on z/VM and is member DHTML in the ERRORS PDS on z/OS.
- ❑ Legacy compound reports are not supported.

Example: Creating a DHTML Report

The following example creates a DHTML file that has an image with absolute positioning:

```
SET HTMLARCHIVE = OFF
TABLE FILE GGSALES
SUM UNITS BY CATEGORY BY PRODUCT
ON TABLE SUBHEAD
"Report on Units Sold"
" "
" "
" "
" "
" "
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD FORMAT DHTML
ON TABLE SET STYLE *
TYPE=TABHEADING,IMAGE=c:\images\GOTHAM.GIF, POSITION=(.25 .25),
 SIZE=(.5 .5), $
ENDSTYLE
END
```

The output shows that the look and positioning are the same as they would be for a PDF report:

```
Report on Units Sold


Category      Product           Unit Sales

Coffee        Capuccino             189217
              Espresso              308986
              Latte                 878063
Food          Biscotti              421377
              Croissant             630054
              Scone                 333414
Gifts         Coffee Grinder        186534
              Coffee Pot            190695
              Mug                   360570
              Thermos               190081
```

Excel Cell Locking

How to:

Enable Spreadsheet Locking

Lock Specific Cells Within a Spreadsheet

Example:

Locking an Entire Excel Spreadsheet

Locking a Single Column on an Excel Spreadsheet

Starting in Version 7.6.5, you can make cells in an Excel spreadsheet read-only.

Using StyleSheet attributes, you can lock Excel spreadsheet values so they are read-only. These attributes apply to all Excel formats including EXL2K, EXL2K PIVOT, and EXL2K FORMULA.

Syntax: How to Enable Spreadsheet Locking

To enable locking, use the following attributes:

```
TYPE=REPORT, PROTECTED={ON|OFF}, [LOCKED={ON|OFF}],$
```

where:

`TYPE=REPORT, PROTECTED=ON`

Is necessary to enable spreadsheet locking. PROTECTED=OFF is the default. If you omit the LOCKED=OFF attribute, the entire spreadsheet is locked.

`LOCKED=ON`

Locks the entire spreadsheet. ON is the default value.

`LOCKED=OFF`

Unlocks the spreadsheet as a whole, but enables you to lock or unlock specific cells or groups of cells.

Syntax: How to Lock Specific Cells Within a Spreadsheet

Once you include the following declaration in your StyleSheet, you can specify the LOCKED attribute for specific cells or groups of cells:

```
TYPE=REPORT, PROTECTED=ON, LOCKED=OFF,$
```

To lock specific parts of the spreadsheet, add the LOCKED=ON attribute to the StyleSheet declaration for the cells you want to lock.

```
TYPE=type, [ COLUMN=columnspec ] ,LOCKED={ON|OFF},$
```

where:

`type`

Is the type of element that describes the cells to be locked.

`columnspec`

Is a valid column specification.

Example: Locking an Entire Excel Spreadsheet

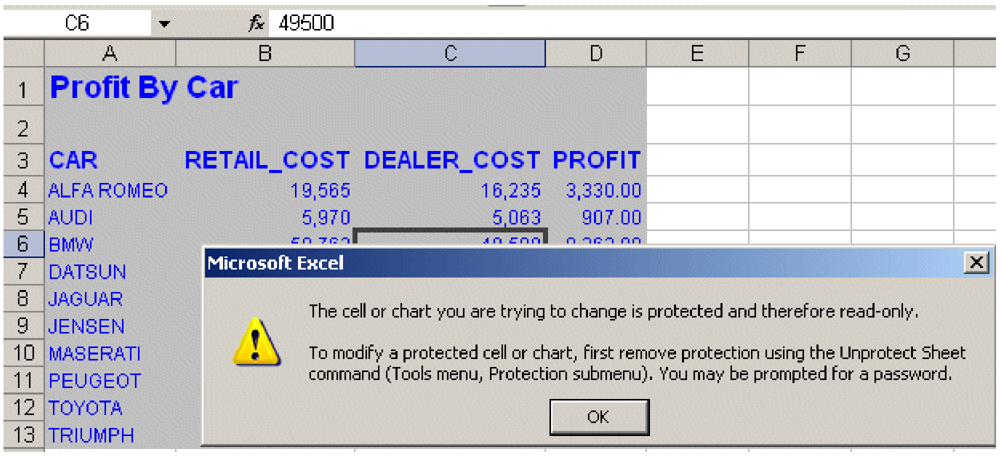
The following request locks the entire spreadsheet because the StyleSheet declarations include the following declaration:

```
TYPE=REPORT, PROTECTED=ON, $
```

The request is:

```
TABLE FILE CAR
HEADING
"Profit By Car "
" "
SUM RETAIL_COST AND DEALER_COST AND
COMPUTE PROFIT/D12.2 = RETAIL_COST - DEALER_COST;
BY CAR
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD AS EXLFORM1 FORMAT EXL2K
ON TABLE SET STYLE *
TYPE=REPORT, COLOR=BLUE, BACKCOLOR=SILVER, SIZE=9,$
TYPE=REPORT, PROTECTED=ON,            $
TYPE=HEADING, STYLE=BOLD, SIZE=14, $
TYPE=TITLE, STYLE=BOLD, SIZE=11,$
ENDSTYLE
END
```

You cannot edit any value on the spreadsheet. Any attempt to do so displays a message that the sheet is protected:


C6 49500

Profit By Car

CAR	RETAIL_COST	DEALER_COST	PROFIT
ALFA ROMEO	19,565	16,235	3,330.00
AUDI	5,970	5,063	907.00
BMW			
DATSUN			
JAGUAR			
JENSEN			
MASERATI			
PEUGEOT			
TOYOTA			
TRIUMPH			

Microsoft Excel

The cell or chart you are trying to change is protected and therefore read-only.

To modify a protected cell or chart, first remove protection using the Unprotect Sheet command (Tools menu, Protection submenu). You may be prompted for a password.

Example: Locking a Single Column on an Excel Spreadsheet

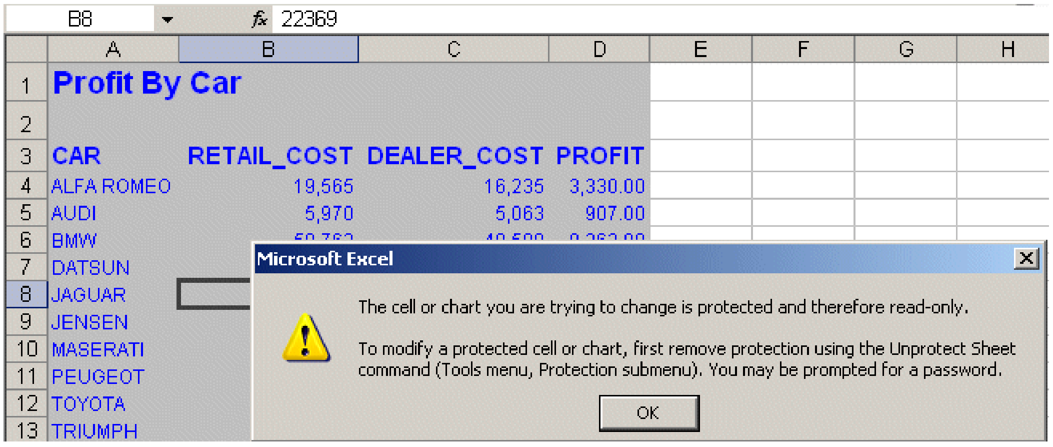
The following request locks the second column (RETAIL_COST) because the StyleSheet declarations include the following declarations

```
TYPE=REPORT, PROTECTED=ON, LOCKED=OFF, $
TYPE=DATA, COLUMN=2, LOCKED=ON,$
```

The request is:

```
TABLE FILE CAR
HEADING
"Profit By Car "
" "
SUM RETAIL_COST AND DEALER_COST AND
COMPUTE PROFIT/D12.2 = RETAIL_COST - DEALER_COST;
BY CAR
ON TABLE SET PAGE-NUM OFF
ON TABLE HOLD AS EXLFORM2 FORMAT EXL2K
ON TABLE SET STYLE *
TYPE=REPORT, COLOR=BLUE, BACKCOLOR=SILVER, SIZE=9,$
TYPE=REPORT, PROTECTED=ON, LOCKED=OFF,$
TYPE=HEADING, STYLE=BOLD, SIZE=14, $
TYPE=TITLE, STYLE=BOLD, SIZE=11,$
TYPE=DATA, COLUMN=2, LOCKED=ON,$
ENDSTYLE
END
```

You cannot edit any value in column 2, although you can edit values in other columns. Any attempt to edit a value in column 2 displays a message that the cells are protected:



B8 22369

	A	B	C	D
1	**Profit By Car**			
2				
3	**CAR**	**RETAIL_COST**	**DEALER_COST**	**PROFIT**
4	ALFA ROMEO	19,565	16,235	3,330.00
5	AUDI	5,970	5,063	907.00
6	BMW			
7	DATSUN			
8	JAGUAR			
9	JENSEN			
10	MASERATI			
11	PEUGEOT			
12	TOYOTA			
13	TRIUMPH			

Microsoft Excel

The cell or chart you are trying to change is protected and therefore read-only.

To modify a protected cell or chart, first remove protection using the Unprotect Sheet command (Tools menu, Protection submenu). You may be prompted for a password.

4 General Enhancements

This chapter describes features that work in multiple FOCUS environments.

Topics:

DB_LOOKUP: Retrieving a Value From a Lookup Data Source

How to:

Retrieve a Value From a Lookup Data Source

Example:

Retrieving a Value From a Fixed Format Sequential File in a TABLE Request

Reference:

Usage Notes for DB_LOOKUP

The DB_LOOKUP function enables you to retrieve a value from one data source when running a request against another data source, without joining or combining the two data sources.

DB_LOOKUP compares pairs of fields from the source and lookup data sources to locate matching records and retrieve the value to return to the request. You can specify as many pairs as needed to get to the lookup record that has the value you want to retrieve. If your field list pairs do not lead to a unique lookup record, the first matching lookup record retrieved is used.

DB_LOOKUP can be called in a DEFINE command, TABLE COMPUTE command, or MODIFY COMPUTE command.

There are no restrictions on the source file. The lookup file can be any non-FOCUS data source that is supported as the cross referenced file in a cluster join. The lookup fields used to find the matching record are subject to the rules regarding cross-referenced join fields for the lookup data source. A fixed format sequential file can be the lookup file if it is sorted in the same order as the source file.

Syntax: How to Retrieve a Value From a Lookup Data Source

```
DB_LOOKUP(look_mf, srcfld1, lookfld1, srcfld2, lookfld2, ..., returnfld);
```

where:

look_mf

Is the lookup Master File.

srcfld1, srcfld2 ...

Are fields from the source file used to locate a matching record in the lookup file.

lookfld1, lookfld2 ...

Are columns from the lookup file that share values with the source fields. Only columns in the table or file can be used; columns created with DEFINE cannot be used. For multi-segment synonyms only columns in the top segment can be used.

returnfld

Is the name of a column in the lookup file whose value is returned from the matching lookup record. Only columns in the table or file can be used; columns created with DEFINE cannot be used.

Reference: Usage Notes for DB_LOOKUP

- The maximum number of pairs that can be used to match records is 63.
- If the lookup file is a fixed format sequential file, it must be sorted and retrieved in the same order as the source file. The sequential file's key field must be the first lookup field specified in the DB_LOOKUP request. If it is not, no records will match.

  In addition, if a DB_LOOKUP request against a sequential file is issued in a DEFINE FILE command, you must clear the DEFINE FILE command at the end of the TABLE request that references it or the lookup file will remain open. It will not be reusable until closed and may cause problems when you exit FOCUS. Other types of lookup files can be reused without clearing the DEFINE. They will be cleared automatically when all DEFINE fields are cleared.
- If the lookup field has the MISSING=ON attribute in its Master File and the DEFINE or COMPUTE command specifies MISSING ON, the missing value is returned when the lookup field is missing. Without MISSING ON in both places, the missing value is converted to a default value (blank for an alphanumeric field, zero for a numeric field).
- Source records display on the report output even if they lack a matching record in the lookup file.
- Only real fields in the lookup Master File are valid as lookup and return fields.
- If there are multiple rows in the lookup table where the source field is equal to the lookup field, the first value of the return field is returned.

Example: Retrieving a Value From a Fixed Format Sequential File in a TABLE Request

The following procedure creates a fixed format sequential file named GSALE from the GGSALES data source. The fields in this file are PRODUCT (product description), CATEGORY (product category), and PCD (product code). The file is sorted on the PCD field:

```
SET ASNAMES = ON
TABLE FILE GGSALES
SUM PRODUCT CATEGORY
BY PCD
ON TABLE HOLD AS GSALE FORMAT ALPHA
END
```

The following Master File is generated as a result of the HOLD command:

```
FILENAME=GSALE, SUFFIX=FIX     , $
  SEGMENT=GSALE, SEGTYPE=S1, $
    FIELDNAME=PCD, ALIAS=E01, USAGE=A04, ACTUAL=A04, $
    FIELDNAME=PRODUCT, ALIAS=E02, USAGE=A16, ACTUAL=A16, $
    FIELDNAME=CATEGORY, ALIAS=E03, USAGE=A11, ACTUAL=A11, $
```

The following TABLE request against the GGPRODS data source, sorts the report on the field that matches the key field in the lookup file. It retrieves the value of the CATEGORY field from the GSALE lookup file by matching on the product code and product description fields. Note that the DEFINE FILE command is cleared at the end of the request:

```
DEFINE FILE GGPRODS
PCAT/A11 MISSING ON = DB_LOOKUP(GSALE,  PRODUCT_ID, PCD,
         PRODUCT_DESCRIPTION, PRODUCT, CATEGORY);
END
TABLE FILE GGPRODS
PRINT PRODUCT_DESCRIPTION PCAT
BY PRODUCT_ID
END
DEFINE FILE GGPRODS CLEAR
END
```

Because the GSALE Master File does not define the CATEGORY field with the MISSING=ON attribute, the PCAT column displays a blank in those rows that have no matching record in the lookup file:

```
Product
Code     Product           PCAT
-------  -------           ----
B141     Hazelnut
B142     French Roast
B144     Kona
F101     Scone             Food
F102     Biscotti          Food
F103     Croissant         Food
G100     Mug               Gifts
G104     Thermos           Gifts
G110     Coffee Grinder    Gifts
G121     Coffee Pot        Gifts
```

If you add the MISSING=ON attribute to the CATEGORY field in the GSALE Master File, the PCAT column displays a missing data symbol in rows that do not have a matching record in the lookup file:

```
Product
Code     Product           PCAT
-------  -------           ----
B141     Hazelnut          .
B142     French Roast      .
B144     Kona              .
F101     Scone             Food
F102     Biscotti          Food
F103     Croissant         Food
G100     Mug               Gifts
G104     Thermos           Gifts
G110     Coffee Grinder    Gifts
G121     Coffee Pot        Gifts
```

HEXTR and HMASK Date-Time Functions

How to:

Extract Multiple Components From a Date-Time Value

Move Multiple Date-Time Components to a Target Date-Time Field

Example:

Extracting Hour and Minute Components Using HEXTR

Changing a Date-Time Field Using HMASK

Reference:

Usage Notes for the HMASK Function

The HEXTR function extracts one or more components from a date-time value and moves them to a target date-time field with all other components set to zero.

The HMASK function extracts one or more components from a date-time value and moves them to a target date-time field with all other components of the target field preserved.

Syntax: How to Extract Multiple Components From a Date-Time Value

```
HEXTR(source, 'componentstring', length, outfield)
```

where:

source

Is the a date-time value from which to extract the specified components.

componentstring

Is a string of codes, in any order, that indicates which components are to be extracted and moved to the output date-time field. The following table shows the valid values. The string is considered to be terminated by any character not in this list:

Code	Description
C	century (the two high-order digits only of the four-digit year)
Y	year (the two low-order digits only of the four-digit year)
YY	Four digit year.
M	month
D	day

Code	Description
H	hour
I	minutes
S	seconds
s	milliseconds (the three high-order digits of the six-digit microseconds value)
u	microseconds (the three low-order digits of the six-digit microseconds value)
m	All six digits of the microseconds value.

length

Is the length of the returned date-time value. Valid values are:

8 - indicates a time value that includes milliseconds.

10 - indicates a time value the includes microseconds.

outfield

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This field must be in date-time format (data type H).

Example: Extracting Hour and Minute Components Using HEXTR

The VIDEOTR2 data source has a date-time field named TRANSDATE of type HYYMDI. The following request selects all records containing the time 09:18AM, regardless of the value of the remaining components:

```
TABLE FILE VIDEOTR2
PRINT TRANSDATE
BY LASTNAME
BY FIRSTNAME
WHERE HEXTR(TRANSDATE, 'HI', 8, 'HYYMDI') EQ DT(09:18AM)
END
```

The output is:

```
LASTNAME         FIRSTNAME   TRANSDATE
--------         ---------   ---------
DIZON            JANET       1999/11/05 09:18
PETERSON         GLEN        1999/09/09 09:18
```

Syntax: How to Move Multiple Date-Time Components to a Target Date-Time Field

```
HMASK(source, 'componentstring', input, length, outfield)
```

where:

source

Is the date-time value from which the specified components are extracted.

componentstring

Is a string of codes, in any order, that indicates which components are to be extracted and moved to the output date-time field. The following table shows the valid values. The string is considered to be terminated by any character not in this list:

Code	Description
C	century (the two high-order digits only of the four-digit year)
Y	year (the two low-order digits only of the four-digit year)
YY	Four digit year.
M	month
D	day
H	hour
I	minutes
S	seconds
s	milliseconds (the three high-order digits of the six-digit microseconds value)
u	microseconds (the three low-order digits of the six-digit microseconds value)
m	All six digits of the microseconds value.

input

Is the date-time value that provides all the components for the output that are not specified in the component string.

length

Is the length of the returned date-time value. Valid values are:

8 - indicates a time value that includes milliseconds.

10 - indicates a time value the includes microseconds.

`outfield`

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This field must be in date-time format (data type H).

Reference: Usage Notes for the HMASK Function

HMASK processing is subject to the DTSTRICT setting. Moving the day (D) component without the month (M) component could lead to an invalid result, which is not permitted if the DTSTRICT setting is ON. Invalid date-time values cause any date-time function to return zeroes.

Example: Changing a Date-Time Field Using HMASK

The VIDEOTRK data source has a date-time field named TRANSDATE of format HYYMDI. The following request changes any TRANSDATE value with a time component greater than 11:00 to 8:30 of the following day. First the HEXTR function extracts the hour and minutes portion of the value and compares it to 11:00. If it is greater than 11:00, the HADD function calls HMASK to change the time to 08:30 and adds one day to the date:

```
DEFINE FILE VIDEOTR2
ORIG_TRANSDATE/HYYMDI = TRANSDATE;
TRANSDATE =
IF HEXTR(TRANSDATE, 'HI', 8, 'HHI') GT DT(12:00)
   THEN HADD (HMASK(DT(08:30), 'HISs', TRANSDATE, 8, 'HYYMDI'), 'DAY',
    1,8, 'HYYMDI')
   ELSE TRANSDATE;
END

TABLE FILE VIDEOTR2
PRINT ORIG_TRANSDATE TRANSDATE
BY LASTNAME
BY FIRSTNAME
WHERE ORIG_TRANSDATE NE TRANSDATE
END
```

The output is:

```
LASTNAME         FIRSTNAME   ORIG_TRANSDATE    TRANSDATE
--------         ---------   --------------    ---------
BERTAL           MARCIA      1999/07/29 12:19  1999/07/30 08:30
GARCIA           JOANN       1998/05/08 12:48  1998/05/09 08:30
                             1999/11/30 12:12  1999/12/01 08:30
PARKER           GLENDA      1999/01/06 12:22  1999/01/07 08:30
RATHER           MICHAEL     1998/02/28 12:33  1998/03/01 08:30
WILSON           KELLY       1999/06/26 12:34  1999/06/27 08:30
```

Specifying Precision for Dialogue Manager Calculations

This setting enables Dialogue Manager -SET commands to display and store accurate numeric variable values without using the FTOA function.

Without this setting, results of numeric calculations are returned as integer numbers, although the calculations themselves employ double-precision arithmetic. To return a number with decimal precision without this setting, you have to enter the calculation as input into subroutine FTOA, where you can specify the number of decimal places returned.

The SET DMPRECISION command gives users the option of either accepting the default truncation of the decimal portion of output from arithmetic calculations, or specifying up to nine decimal places for rounding.

Syntax: How to Specify Precision for Dialogue Manager Calculations

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

```
SET DMPRECISION = {OFF|n}
```

where:

OFF

Specifies truncation without rounding after the decimal point. OFF is the default value.

n

Is a positive number from 0-9, indicating the point of rounding. Note that n=0 results in a rounded integer value.

Example: Rounding Using DMPRECISION

The following table below shows the result of dividing 20 by 3 with varying DMPRECISION (DMP) settings:

SET DMPRECISION =	Result
OFF	6
0	7
1	6.7
2	6.67
9	6.666666667

Reference: Usage Notes for SET DMPRECISION

- When using SET DMPRECISION, you must include -RUN after the SET DMPRECISION command to ensure that it is set prior to any numeric -SET commands.
- As the actual conversion to double precision follows the rules for the operating system, the values may vary from platform to platform.

Displaying SET Parameters by Functional Area

How to:

Display SET Parameter Values Categorized by Functional Areas

Display SET Parameter Values for a Specific Functional Area

Display the List of Categories

Example:

Viewing Parameters by Functional Category

The ? SET BY CATEGORY query allows users to display settable parameter values grouped by major functional categories.

Syntax: How to Display SET Parameter Values Categorized by Functional Areas

Issue the following command in any supported profile, or in a focexec, or at the command prompt, to display settable parameter values by functional area :

```
? SET BY CATEGORY
```

The functional areas available for display are listed below.

MEMORY	Options that affect size of memory used.
DATES	Options that control date input/output.
SECURITY	Security options.
POOLTABLE	Options relevant only to POOLTABLE.
SINK	Options relevant only to SINK MACHINEs.
SEND	SEND command parameters.
COMPUTATION	Options that affect computations.
MDI	MDI parameters.
EXTERNALSORT	External sort parameters.
FOCCALC	FOCCALC environmental parameters.
ENVIRONMENT	General working environment options.
WEBFOCUS	WEBFOCUS environmental parameters.
REPORT	Options that affect report appearance.
GRAPH	Classical GRAPH control parameters.
STYLESHEET	Options that affect STYLESHEET.
RETRIEVAL	Parameters that affect data retrieval.
HOLD	Options that affect HOLD output.
PLATFORM	Platform-dependent options.
MAINFRAME	Options relevant only to IBM/MAINFRAME.
MSWINDOWS	Options relevant only to MSWINDOWS

Example: Viewing Parameters by Functional Category

To view the current values for parameters in all categories, enter:

```
? SET BY CATEGORY
```

This displays the existing parameter values in each category (first four of twenty shown).

```
                         MEMORY UTILIZATION PARAMETERS

BINS                     64    CACHE                     0    CALCMEMORY               5
LOADLIMIT                64    MDIBINS                8000    POOLMEMORY           16384
XFBINS      16 (passive)

                            DATE CONTROL PARAMETERS

ALLOWCVTERR             OFF    BUSDAYS             _MTWTF_    DATEDISPLAY            OFF
DATEFNS                  ON    DATEFORMAT              MDY    DATETIME     STARTUP/RESET
DEFCENT                  19    DTSTANDARD              OFF    DTSTRICT                ON
HDAY                           TESTDATE              TODAY    WEEKFIRST                7
YRTHRESH                  0

                        SECURITY ENVIRONMENT PARAMETERS

PASS                 ??????    PERMPASS             ??????    SUSI                   OFF

                          POOLTABLE RELATED PARAMETERS
DEJAVU                 ????    ESTLINES                  0    ESTRECORDS               0
HRATIO                 ????    MAXADRTABLE            ????    MAXEXTSRTS            ????
MAXMNM                 ????    MAXMRGSTRNGS           ????    MAXPOOLMEM           32768
MAXSORTS               ????    MINADRTABLE            ????    MINEXTVSPACE          ????
MINMTI                 ????    MINMTX                 ????    MTXFDG                ????
MXMFOC                 ????    POOL                    OFF    POOLBATCH              OFF
POOLFEATURE             OFF    POOLMEMORY            16384    POOLORDER             ????
POOLRESERVE            1024    PTBDBG                 ????    PTDFCORE              ????
ROUNDR                 ????    SURPRI                 ????    THRSHF                ????
THRSHX                 ????    TRUST1                 ????
```

Syntax: How to Display SET Parameter Values for a Specific Functional Area

Issue the following commands in any supported profile, in a focexec, or at the command prompt, to display settable parameter values for a specified functional area:

```
? SET CATEGORY categoryname
```

where:

categoryname

Is one of the categories.

Syntax: How to Display the List of Categories

Issue the following command in any supported profile, in a focexec, or at the command prompt, to display settable parameter values for a specified functional area:

```
? SET CATEGORY HELP
```

New QUOTEP Option for Continental Decimal Notation

How to:

Specify Continental Decimal Notation

Example:

Setting CDN Numeric Notation to Apostrophes and Periods

Reference:

Usage Notes for Continental Decimal Notation

You can use the SET CDN command to specify the characters used as decimal and thousands separators for numbers displayed on report output.

Syntax: How to Specify Continental Decimal Notation

```
SET CDN = option
```

```
ON TABLE SET CDN option
```

where:

option

Can be one of the following:

OFF

Turns CDN off. For example, the number 3,045,000.76 is represented as 3,045,000.76. OFF should be used for the USA, Canada, Mexico, and the United Kingdom. OFF is the default (standard) value.

ON

Designates the decimal separator as a comma and the thousands separator as a period. For example, the number 3,045,000.76 is represented as 3.045.000,76. 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. For example, the number 3,045,000.76 is represented as 3 045 000,76. 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. For example, the number 3,045,000.76 is represented as 3'045'000,76. QUOTE should be used for Switzerland.

`QUOTEP`

Sets the decimal point as a period and the thousands separator as an apostrophe. For example, the number 3,045,000.76 is represented as 3'045'000.76.

Reference: Usage Notes for Continental Decimal Notation

If the display format of a report is Excel 2000 or later, Continental Decimal Notation is controlled by the settings on the user's computer. That is, numbers in report output are formatted according to the convention of the locale (location) set in regional or browser language options.

Example: Setting CDN Numeric Notation to Apostrophes and Periods

```
SET CDN=QUOTEP
TABLE FILE EMPLOYEE
PRINT FIRST_NAME LAST_NAME SALARY
END
```

The output is:

```
FIRST_NAME  LAST_NAME                SALARY
----------  ---------                ------
ALFRED      STEVENS              $11'000.00
ALFRED      STEVENS              $10'000.00
MARY        SMITH                $13'200.00
DIANE       JONES                $18'480.00
DIANE       JONES                $17'750.00
RICHARD     SMITH                 $9'500.00
RICHARD     SMITH                 $9'050.00
JOHN        BANNING              $29'700.00
JOAN        IRVING               $26'862.00
JOAN        IRVING               $24'420.00
ANTHONY     ROMANS               $21'120.00
JOHN        MCCOY                $18'480.00
ROSEMARIE   BLACKWOOD            $21'780.00
ROGER       MCKNIGHT             $16'100.00
ROGER       MCKNIGHT             $15'000.00
MARY        GREENSPAN             $9'000.00
MARY        GREENSPAN             $8'650.00
```

Establishing a Default Value for the &ECHO Variable

How to:

Set a Default Value for &ECHO

Example:

Setting a Default Value for the &ECHO Variable

The Dialogue Manager variable &ECHO controls whether commands are displayed as they execute.

By default, &ECHO is set to OFF in every procedure in an application, which means that commands are not displayed as they execute. You can change this value for a specific procedure by specifying a value on the EX command, in a -SET command, or in a -DEFAULT command.

Even if a calling procedure is executed with &ECHO=ALL, any called procedure that does not explicitly set the value of &ECHO is executed using the default value of OFF.

The SET DEFECHO command enables you to specify a default value for &ECHO that spans all procedures executed in an application or session. Any FOCEXEC executed that does not explicitly establish a value for &ECHO uses the default value established by the SET DEFECHO command. You can explicitly control the value of &ECHO for an individual procedure by either passing a value on the EX command, issuing a -SET command for &ECHO, or issuing a -DEFAULT command for &ECHO.

Syntax: How to Set a Default Value for &ECHO

```
SET DEFECHO = {OFF|ON|ALL}
```

where:

`OFF`

Establishes OFF as the default value for &ECHO. OFF is the default value.

`ON`

Establishes ON as the default value for &ECHO. ON displays FOCUS commands that are expanded and stacked for execution.

`ALL`

Establishes ALL as the default value for &ECHO. ALL displays Dialogue Manager commands and FOCUS commands that are expanded and stacked for execution.

Example: Setting a Default Value for the &ECHO Variable

The following procedure executes two FOCEXECs. The first FOCEXEC, RJUST1, right justifies the last name of employees in the EMPLOYEE data source, and the second FOCEXEC, CTRFLD1, centers the names. The first EX command sets &ECHO to ON:

```
EX RJUST1 ECHO=ON
EX CTRFLD1
```

The RJUST1 procedure is echoed before the report output is generated. The CTRFLD1 procedure uses the default value, OFF, for &ECHO, so the procedure is not echoed.

Echo for RJUST1:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
RIGHT_NAME/A15 = RJUST(15, LAST_NAME, RIGHT_NAME);
WHERE DEPARTMENT EQ 'MIS';
END
```

Report output for RJUST1:

```
LAST_NAME        RIGHT_NAME
---------        ----------
SMITH                      SMITH
JONES                      JONES
MCCOY                      MCCOY
BLACKWOOD              BLACKWOOD
GREENSPAN              GREENSPAN
CROSS                      CROSS
```

Report output for CTRFLD1:

```
LAST_NAME        CENTER_NAME
---------        -----------
SMITH               SMITH
JONES               JONES
MCCOY               MCCOY
BLACKWOOD         BLACKWOOD
GREENSPAN         GREENSPAN
CROSS               CROSS
```

Running the procedure with SET DEFECHO=ON, but specifying ECHO=OFF on the EX command for RJUST1, causes the CTRFLD1 procedure to use the default of ON. However, the RJUST1 procedure uses the specified value of OFF:

```
SET DEFECHO = ON
EX RJUST1 ECHO=OFF
EX CTRFLD1
```

Report output for RJUST1:

```
LAST_NAME        RIGHT_NAME
---------        ----------
SMITH                      SMITH
JONES                      JONES
MCCOY                      MCCOY
BLACKWOOD              BLACKWOOD
GREENSPAN              GREENSPAN
CROSS                      CROSS
```

Echo for CTRFLD1:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
CENTER_NAME/A12 = CTRFLD(LAST_NAME, 12, 'A12');
WHERE DEPARTMENT EQ 'MIS'
END
```

Report output for CTRFLD1:

```
LAST_NAME        CENTER_NAME
---------        -----------
SMITH               SMITH
JONES               JONES
MCCOY               MCCOY
BLACKWOOD         BLACKWOOD
GREENSPAN         GREENSPAN
CROSS               CROSS
```

Retrieving the Site Code of the Connected User

How to:

Retrieve the Site Code

Example:

Querying the Site Code

The FOCUS site code is installed as part of the License Management facility.

Once the site code has been installed, you can retrieve its value by issuing the ? SITECODE query command. If the site code has not been installed, you will get a message indicating that the site code is not available.

Syntax: How to Retrieve the Site Code

```
? SITECODE
```

Example: Querying the Site Code

Assume you installed the License Management facility with site code A52709b.

Issue the following query command:

```
? SITECODE
```

The output is:

```
SITE CODE A52709b
```

If the site code is not installed, the ? SITECODE query returns the following message:

```
SITE CODE NOT AVAILABLE
```

Establishing a Non-Overridable User Password

How to:

Set a Non-Overridable User Password

Reference:

Usage Notes for Non-Overridable User Passwords

Example:

Setting a Non-Overridable User Password

The PERMPASS parameter establishes a user password that remains in effect throughout a session or connection. You can issue this setting in any supported profile but is most useful when established for an individual user by setting it in a user profile. It cannot be set in an ON TABLE phrase. It is recommended that it not be set in FOCPARM or FOCPROF because it would then apply to all users. In a FOCUS session, SET PERMPASS can be issued in PROFILE, a FOCEXEC, or at the command prompt.

All security rules established in the DBA sections of existing Master Files are respected when PERMPASS is in effect. The user cannot issue the SET PASS or SET USER command to change to a user password with different security rules. Any attempt to do so generates the following message:

```
permanent PASS is in effect. Your PASS will not be honored.
VALUE WAS NOT CHANGED
```

Only one permanent password can be established in a session. Once it is set, it cannot be changed within the session.

Syntax: How to Set a Non-Overridable User Password

```
SET PERMPASS=userpass
```

where:

userpass

Is the user password used for all access to data sources with DBA security rules established in their associated Master Files.

Example: Setting a Non-Overridable User Password

Consider the MOVIES Master File with the following DBA rules in effect:

```
DBA=USER1,$
USER = USERR,  ACCESS = R ,$
USER = USERU,  ACCESS = U ,$
USER = USERW,  ACCESS = W ,$
USER = USERRW, ACCESS = RW,$
```

The following FOCEXEC sets a permanent password:

```
SET PERMPASS = USERU
TABLE FILE MOVIES
PRINT TITLE BY DIRECTOR
END
```

The user has ACCESS=U and, therefore, is not allowed to issue a table request against the file:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE:
CAR
BYPASSING TO END OF COMMAND
```

The permanent password cannot be changed:

```
SET PERMPASS = USERRW
```

```
permanent PASS is in effect. Your PASS will not be honored.
VALUE WAS NOT CHANGED
```

The user password cannot be changed:

```
SET PASS = USERRW
```

```
permanent PASS is in effect. Your PASS will not be honored.
VALUE WAS NOT CHANGED
```

Reference: Usage Notes for Non-Overridable User Passwords

If you use FOCUSID to set passwords externally, these external passwords may be overridable or non-overridable. If they are non-overridable, they take precedence over the PERMPASS setting.

SET PERMPASS is supported in MSO. The profiles available in order of precedence are:

- ❏ Member PROFILE in the data set allocated to ddname MSOPROF. This profile is executed for all users.
- ❏ Members with users' user IDs as names in the data set allocated to ddname MSOPROF. The profile is executed for the user corresponding to the user ID.

- Member PROFILE in the data set allocated by a user to ddname FOCEXEC. This profile is executed for that specific user and is a standard FOCUS profile.
- Member SHELPROF in the data set allocated to ddname FOCEXEC in the MSO startup JCL. This profile is executed for all users and is a standard FOCUS profile.

Creating a Standard Quote-Delimited String

How to:

Create a Standard Quote-Delimited Character String

Reference:

Usage Notes for Quote-Delimited Character Strings

Example:

Creating a Standard Quote-Delimited Character String

Converting User Input to a Standard Quote-Delimited Character String

Using Quote-Delimited Strings With Relational Data Adapters

Character strings must be enclosed in single quotation marks to be handled by most database engines. In addition, embedded single quotation marks are indicated by two contiguous single quotation marks. FOCUS, WebFOCUS, and iWay require quotes around variables containing delimiters, which include spaces and commas.

The QUOTEDSTRING suffix on a Dialogue Manager variable applies the following two conversions to the contents of the variable:

- Any single quotation mark embedded within a string is converted to two single quotation marks.
- Single quotation marks are added around the string.

Dialogue Manager commands differ in their ability to handle character strings that are not enclosed in single quotation marks and contain embedded blanks. An explicit or implied -PROMPT command can read such a string. The entire input string is then enclosed in single quotation marks when operated on by .QUOTEDSTRING.

Note: When using the -SET command to reference a character string, ensure the character string is enclosed in single quotes to prevent errors.

Syntax: How to Create a Standard Quote-Delimited Character String

```
&var.QUOTEDSTRING
```

where:

&var

Is a Dialogue Manager variable.

Example: Creating a Standard Quote-Delimited Character String

The following example shows the results of the QUOTEDSTRING suffix on input strings.

```
-SET &A = ABC;
-SET &B = 'ABC';
-SET &C = O'BRIEN;
-SET &D = 'O'BRIEN';
-SET &E = 'O''BRIEN';
-SET &F = O''BRIEN;
-SET &G = OBRIEN';
-TYPE  ORIGINAL = &A QUOTED = &A.QUOTEDSTRING
-TYPE  ORIGINAL = &B QUOTED = &B.QUOTEDSTRING
-TYPE  ORIGINAL = &C QUOTED = &C.QUOTEDSTRING
-TYPE  ORIGINAL = &D QUOTED = &D.QUOTEDSTRING
-TYPE  ORIGINAL = &E QUOTED = &E.QUOTEDSTRING
-TYPE  ORIGINAL = &F QUOTED = &F.QUOTEDSTRING
-TYPE  ORIGINAL = &G QUOTED = &G.QUOTEDSTRING
```

The output is:

```
ORIGINAL = ABC        QUOTED = 'ABC'
ORIGINAL = ABC        QUOTED = 'ABC'
ORIGINAL = O'BRIEN    QUOTED = 'O''BRIEN'
ORIGINAL = O'BRIEN    QUOTED = 'O''BRIEN'
ORIGINAL = O'BRIEN    QUOTED = 'O''BRIEN'
ORIGINAL = O''BRIEN   QUOTED = 'O''''BRIEN'
ORIGINAL = OBRIEN'    QUOTED = 'OBRIEN'''
```

Note: The -SET command will remove single quotes around a string. Notice in the example above that the result of -SET &B = 'ABC' was changed to ORIGINAL = ABC (as shown in the output), prior to the QUOTEDSTRING conversion.

Example: Converting User Input to a Standard Quote-Delimited Character String

The following -TYPE command accepts quoted or unquoted input and displays quoted output.

```
-TYPE THE QUOTED VALUE IS: &E.QUOTEDSTRING
```

The output is:

```
PLEASE SUPPLY VALUES REQUESTED

E=
O'BRIEN
THE QUOTED VALUE IS: 'O''BRIEN'
```

Example: Using Quote-Delimited Strings With Relational Data Adapters

The following procedure creates an Oracle table named SQLVID from the VIDEOTRK data source.

```
TABLE FILE VIDEOTRK
SUM CUSTID EXPDATE PHONE STREET CITY STATE ZIP
   TRANSDATE PRODCODE TRANSCODE QUANTITY TRANSTOT
BY LASTNAME BY FIRSTNAME
WHERE LASTNAME NE 'NON-MEMBER'
ON TABLE HOLD
END
-RUN
CREATE FILE SQLVID
-RUN
MODIFY FILE SQLVID
FIXFORM FROM HOLD
DATA ON HOLD
END
```

Consider the following SQL Translator request:

```
SET TRACEUSER = ON
SET TRACEON = STMTRACE//CLIENT
SQL
SELECT *
FROM SQLVID WHERE LASTNAME  = &1.QUOTEDSTRING;
END
```

When this request is executed, you must enter a last name, in this case O’BRIEN:

```
PLEASE SUPPLY VALUES REQUESTED

1=

O'BRIEN
```

In the generated SQL request, the character string used for the comparison is correctly enclosed in single quotation marks, and the embedded single quote is doubled:

```
SELECT SQLCOR01.CIN , SQLCOR01.LN , SQLCOR01.FN ,
SQLCOR01.EXDAT , SQLCOR01.TEL , SQLCOR01.STR , SQLCOR01.CITY ,
SQLCOR01.PROV , SQLCOR01.POSTAL_CODE , SQLCOR01.OUTDATE ,
SQLCOR01.PCOD , SQLCOR01.TCOD , SQLCOR01.NO , SQLCOR01.TTOT
FROM SQLVID SQLCOR01  WHERE SQLCOR01.LN  = 'O''BRIEN';
```

The output is:

```
CIN    LN                FN      ...
---    --                --      ...
5564   O'BRIEN           DONALD  ...
```

The following input variations are translated to the correct form of quoted string demonstrated in the trace.

```
'O'BRIEN'
'O''BRIEN'
```

Any other variation results in:

- ❏ A valid string that does not match the database value and does not return any rows. For example, O'''BRIEN becomes 'O''''''BRIEN' in the WHERE predicate.
- ❏ An invalid string that produces one of the following messages:

  Error - Semi-colon or END expected

  Error - Missing or Misplaced quotes

  Error - (value entered) is not a valid column

  Error - Syntax error on line ... Unbalanced quotes

Strings without embedded single quotation marks can be entered without quotes or embedded in single quotation marks, either SMITH or 'SMITH'.

If you use &1 without the QUOTEDSTRING suffix in the request, acceptable input strings that retrieve O'Brien's record are:

```
'''O'''BRIEN'''
'''O''''BRIEN'''
```

Using &1 without the QUITEDSTRING suffix, the acceptable form of a string without embedded single quotation marks is '''SMITH'''.

To make a string enclosed in single quotation marks acceptable without the QUOTEDSTRING suffix, use '&1' in the request. In this case, in order to retrieve O'Brien's record, you must enter the string that would have resulted from the QUOTEDSTRING suffix:

```
'O''''BRIEN'
```

To enter a string without embedded single quotation marks using '&1', you can either omit the surrounding single quotation marks or include them: SMITH or 'SMITH'.

Note: The form '&1.QUOTEDSTRING' is not supported.

Reference: Usage Notes for Quote-Delimited Character Strings

❏ An unmatched single quotation mark at the beginning of a character string is treated as invalid input and generates the following message:

```
(FOC257) MISSING QUOTE MARKS:  value;
```

Assignment Between Text Fields and Alphanumeric Fields

How to:

Assign a Value to a Text Field or Assign a Text Field to a Text or Alphanumeric Field

Reference:

Usage Notes for Text Fields in COMPUTE and DEFINE

Example:

Assigning the Result of an Alphanumeric Expression to a Text Field

Both text and alphanumeric fields can be assigned values stored in text fields or alphanumeric expressions in TABLE COMPUTE, MODIFY COMPUTE, and DEFINE commands. If an alphanumeric field is assigned the value of a text field that is too long for the alphanumeric field, the value is truncated before being assigned to the alphanumeric field.

Syntax: How to Assign a Value to a Text Field or Assign a Text Field to a Text or Alphanumeric Field

In a DEFINE FILE command, use the following syntax

```
DEFINE
field/{TXn|An} = {alphaexpression|textfield};
END
```

In a TABLE COMPUTE or MODIFY COMPUTE command, use the following syntax

```
COMPUTE field/{TXn|An} = {alphaexpression|textfield};
```

where:

field

Is a text or alphanumeric field that will receive the text field value or result of the alphanumeric expression.

n

Is the length of the alphanumeric field or the output display length of the text field that will receive the value from the right hand side of the COMPUTE or DEFINE command.

alphaexpression

Is the name of an alphanumeric field, an alphanumeric literal, or an alphanumeric expression.

textfield

Is the name of a text field.

Example: Assigning the Result of an Alphanumeric Expression to a Text Field

This example uses the COURSES data source, which contains a text field, to create an alphanumeric field named ADESC, which truncates the text field at 36 characters, and a new text field named NEWDESC, which is a text version of ADESC:

```
DEFINE FILE COURSES
ADESC/A36    = DESCRIPTION;
NEWDESC/TX36 = ADESC;
END

TABLE FILE COURSES
PRINT ADESC NEWDESC
END
```

The output is:

```
ADESC                                 NEWDESC
-----                                 -------
This course provides the DP professi  This course provides the DP professi
Anyone responsible for designing FOC  Anyone responsible for designing FOC
This is a course in FOCUS efficienci  This is a course in FOCUS efficienci
```

Reference: Usage Notes for Text Fields in COMPUTE and DEFINE

COMPUTE commands in Maintain do not support text fields.

DATETRAN Function

How to:

Format Dates in International Formats

Reference:

Usage Notes for the DATETRAN Function

Example:

Using the DATETRAN Function

The DATETRAN function formats dates in international formats.

Syntax: How to Format Dates in International Formats

```
DATETRAN (indate, '(intype)', '([formatops])', 'lang', outlen, output)
```

where:

indate

Is the input date (in date format) to be formatted. Note that the date format cannot be an alphanumeric or numeric format with date display options (legacy date format).

intype

Is one of the following character strings indicating the input date components and the order in which you want them to display, enclosed in parentheses and single quotation marks:

Single Component Input Type	Description
`'(W)'`	Day of week component only (original format must have only W component).
`'(M)'`	Month component only (original format must have only M component).

Two-Component Input Type	**Description**
`'(YYM)'`	Four-digit year followed by month.
`'(YM)'`	Two-digit year followed by month.
`'(MYY)'`	Month component followed by four-digit year.
`'(MY)'`	Month component followed by two-digit year.

Three- Component Input Type	**Description**
`'(YYMD)'`	Four-digit year followed by month followed by day.
`'(YMD)'`	Two-digit year followed by month followed by day.
`'(DMYY)'`	Day component followed by month followed by four-digit year.
`'(DMY)'`	Day component followed by month followed by two-digit year.
`'(MDYY)'`	Month component followed by day followed by four-digit year.
`'(MDY)'`	Month component followed by day followed by two-digit year.
`'(MD)'`	Month component followed by day (derived from three-component date by ignoring year component).
`'(DM)'`	Day component followed by month (derived from three-component date by ignoring year component).

`formatops`

Is a string of zero or more formatting options enclosed in parentheses and single quotation marks. The parentheses and quotation marks are required even if you do not specify formatting options. Formatting options fall into the following categories:

❏ Options for suppressing initial zeros in month or day numbers.

❏ Options for translating month or day components to full or abbreviated uppercase or default case (mixed case or lowercase depending on the language) names.

❏ Date delimiter options and options for punctuating a date with commas.

Valid options for suppressing initial zeros in month or day numbers are:

Format Option	Description
`m`	Zero-suppresses months (displays numeric months before October as 1 through 9 rather than 01 through 09).
`d`	Displays days before the tenth of the month as 1 through 9 rather than 01 through 09.
`dp`	Displays days before the tenth of the month as 1 through 9 rather than 01 through 09 with a period after the number.
`do`	Displays days before the tenth of the month as 1 through 9. For English (langcode EN) only, displays an ordinal suffix (st, nd, rd, or th) after the number.

Valid month and day name translation options are:

Format Option	Description
`T`	Displays month as an abbreviated name with no punctuation, all uppercase.
`TR`	Displays month as a full name, all uppercase.
`Tp`	Displays month as an abbreviated name followed by a period, all uppercase.
`t`	Displays month as an abbreviated name with no punctuation. The name is all lowercase or initial uppercase, depending on language code.
`tr`	Displays month as a full name. The name is all lowercase or initial uppercase, depending on language code.
`tp`	Displays month as an abbreviated name followed by a period. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`W`	Includes an abbreviated day of the week name at the start of the displayed date, all uppercase with no punctuation.
`WR`	Includes a full day of the week name at the start of the displayed date, all uppercase.

Format Option	Description
`Wp`	Includes an abbreviated day of the week name at the start of the displayed date, all uppercase, followed by a period.
`w`	Includes an abbreviated day of the week name at the start of the displayed date with no punctuation. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`wr`	Includes a full day of the week name at the start of the displayed date. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`wp`	Includes an abbreviated day of the week name at the start of the displayed date followed by a period. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`X`	Includes an abbreviated day of the week name at the end of the displayed date, all uppercase with no punctuation.
`XR`	Includes a full day of the week name at the end of the displayed date, all uppercase.
`Xp`	Includes an abbreviated day of the week name at the end of the displayed date, all uppercase, followed by a period.
`x`	Includes an abbreviated day of the week name at the end of the displayed date with no punctuation. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`xr`	Includes a full day of the week name at the end of the displayed date. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`xp`	Includes an abbreviated day of the week name at the end of the displayed date followed by a period. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).

Valid date delimiter options are:

Format Option	Description
`B`	Uses a blank as the component delimiter. This is the default if the month or day of week is translated or if comma is used.
`.`	Uses a period as the component delimiter.
`-`	Uses a minus sign as the component delimiter. This is the default when the conditions for a blank default delimiter are not satisfied.
`/`	Uses a slash as the component delimiter.
`|`	Omits component delimiters.
`K`	Uses appropriate Asian characters as component delimiters.
`c`	Places a comma after the month name (following T, Tp, TR, t, tp, or tr). Places a comma and blank after the day name (following W, Wp, WR, w, wp, or wr). Places a comma and blank before the day name (following X, XR, x, or xr).
`e`	Displays the Spanish or Portuguese word de or DE between the day and month and between the month and year. The case of the word de is determined by the case of the month name. If the month is displayed in uppercase, DE is displayed; otherwise de is displayed. Useful for formats DMY, DMYY, MY, and MYY.
`D`	Inserts a comma after the day number and before the general delimiter character specified.
`Y`	Inserts a comma after the year and before the general delimiter character specified.

`lang`

Is the two-character standard ISO code for the language into which the date should be translated, enclosed in single quotation marks. Valid language codes are:

'DE'	German
'DU'	Dutch
'EN'	English
'ES'	Spanish
'FR'	French
'GR'	Greek
'HE'	Hebrew
'JA'	Japanese
'NO'	Norwegian (bokmål)
'PT'	Portuguese
'SV'	Swedish
'ZH'	Simplified Chinese

`outlen`

Numeric

Is the length of the output field in bytes. If the length is insufficient, an all blank result is returned. If the length is greater than required, the field is padded with blanks on the right.

`output`

Alphanumeric

Is the name of the field that contains the translated date, or its format enclosed in single quotation marks.

Reference: Usage Notes for the DATETRAN Function

- ❏ The output field, though it must be type A and not AnV, may in fact contain variable length information, since the lengths of month names and day names can vary, and also month and day numbers may be either one or two bytes long if a zero-suppression option is chosen. Unused bytes are filled with blanks.
- ❏ All invalid and inconsistent inputs result in all blank output strings. Missing data also results in blank output.
- ❏ The base dates (1900-12-31 and 1900-12 or 1901-01) are treated as though the DATEDISPLAY setting were ON (that is, not automatically shown as blanks). To suppress the printing of base dates, which have an internal integer value of 0, test for 0 before calling DATETRAN. For example:

```
RESULT/A40 = IF DATE EQ 0 THEN ' ' ELSE
                DATETRAN (DATE, '(YYMD)', '(.t)', 'FR', 40, 'A40');
```

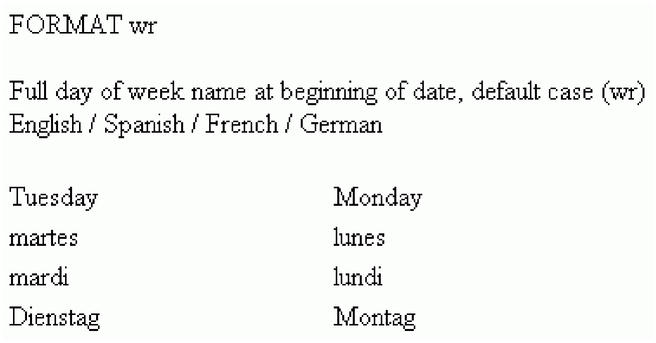
- ❏ Valid translated date components are contained in files named DTLNG*lng* where *lng* is a three-character code that specifies the language. These files must be accessible for each language into which you want to translate dates.
- ❏ If you use a terminal emulator program, it must be set to use a code page that can display the accent marks and characters in the translated dates. You may not be able to display dates translated into European and Asian characters at the same time. Similarly, if you want to print the translated dates, your printer must be capable of printing the required characters.
- ❏ The DATETRAN function is not supported in Dialogue Manager.

Example: Using the DATETRAN Function

The following request prints the day of the week in the default case of the specific language:

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20051003;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT1A/A8=DATETRAN(DATEW, '(W)', '(wr)', 'EN', 8 , 'A8') ;
OUT1B/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'EN', 8 , 'A8') ;
OUT1C/A8=DATETRAN(DATEW, '(W)', '(wr)', 'ES', 8 , 'A8') ;
OUT1D/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'ES', 8 , 'A8') ;
OUT1E/A8=DATETRAN(DATEW, '(W)', '(wr)', 'FR', 8 , 'A8') ;
OUT1F/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'FR', 8 , 'A8') ;
OUT1G/A8=DATETRAN(DATEW, '(W)', '(wr)', 'DE', 8 , 'A8') ;
OUT1H/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'DE', 8 , 'A8') ;
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT wr"
""
"Full day of week name at beginning of date, default case (wr)"
"English / Spanish / French / German"
""
SUM OUT1A AS '' OUT1B AS '' TRANSDATE NOPRINT
OVER OUT1C AS '' OUT1D AS ''
OVER OUT1E AS '' OUT1F AS ''
OVER OUT1G AS '' OUT1H AS ''
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
ON TABLE SET STYLE *
GRID=OFF, $
END
```

The output is:

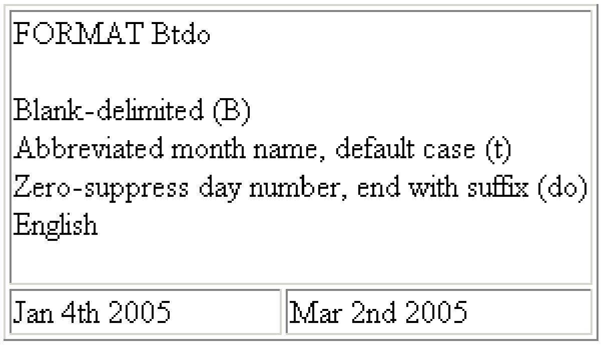


```
FORMAT wr

Full day of week name at beginning of date, default case (wr)
English / Spanish / French / German

Tuesday                    Monday
martes                     lunes
mardi                      lundi
Dienstag                   Montag
```

The following request prints a blank delimited date with an abbreviated month name in English. Initial zeros in the day number are suppressed, and a suffix is added to the end of the number:

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1      ;
DATEW2/W=TRANS2     ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT2A/A15=DATETRAN(DATEYYMD,  '(MDYY)', '(Btdo)', 'EN', 15, 'A15') ;
OUT2B/A15=DATETRAN(DATEYYMD2, '(MDYY)', '(Btdo)', 'EN', 15, 'A15') ;
END
TABLE FILE VIDEOTRK
HEADING
"FORMAT Btdo"
""
"Blank-delimited (B)"
"Abbreviated month name, default case (t)"
"Zero-suppress day number, end with suffix (do)"
"English"
""
SUM OUT2A AS '' OUT2B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



FORMAT Btdo

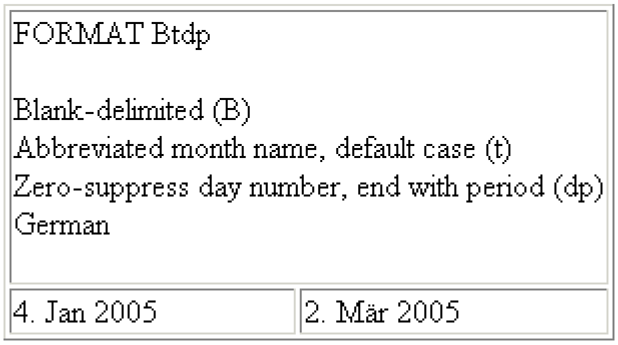
Blank-delimited (B)
Abbreviated month name, default case (t)
Zero-suppress day number, end with suffix (do)
English

Jan 4th 2005	Mar 2nd 2005

The following request prints a blank delimited date with an abbreviated month name in German. Initial zeros in the day number are suppressed, and a period is added to the end of the number:

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT3A/A12=DATETRAN(DATEYYMD,  '(DMYY)', '(Btdp)', 'DE', 12, 'A12');
OUT3B/A12=DATETRAN(DATEYYMD2, '(DMYY)', '(Btdp)', 'DE', 12, 'A12');
END
TABLE FILE VIDEOTRK
HEADING
"FORMAT Btdp"
""
"Blank-delimited (B)"
"Abbreviated month name, default case (t)"
"Zero-suppress day number, end with period (dp)"
"German"
""
SUM OUT3A AS '' OUT3B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:

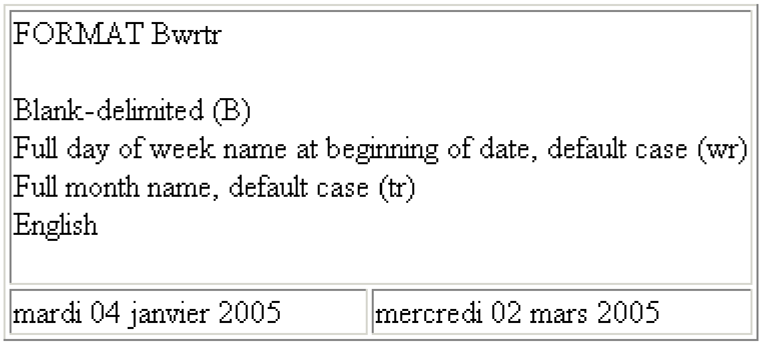


```
FORMAT Btdp

Blank-delimited (B)
Abbreviated month name, default case (t)
Zero-suppress day number, end with period (dp)
German

4. Jan 2005          2. Mär 2005
```

The following request prints a blank delimited date in French with a full day name at the beginning and a full month name, in lower case (the default for French):

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1    ;
DATEYYMD2/YYMDW=TRANS2   ;

OUT4A/A30 = DATETRAN(DATEYYMD,  '(DMYY)', '(Bwrtr)', 'FR', 30, 'A30');
OUT4B/A30 = DATETRAN(DATEYYMD2, '(DMYY)', '(Bwrtr)', 'FR', 30, 'A30');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Bwrtr"
""
"Blank-delimited (B)"
"Full day of week name at beginning of date, default case (wr)"
"Full month name, default case (tr)"
"English"
""
SUM OUT4A AS '' OUT4B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



FORMAT Bwrtr

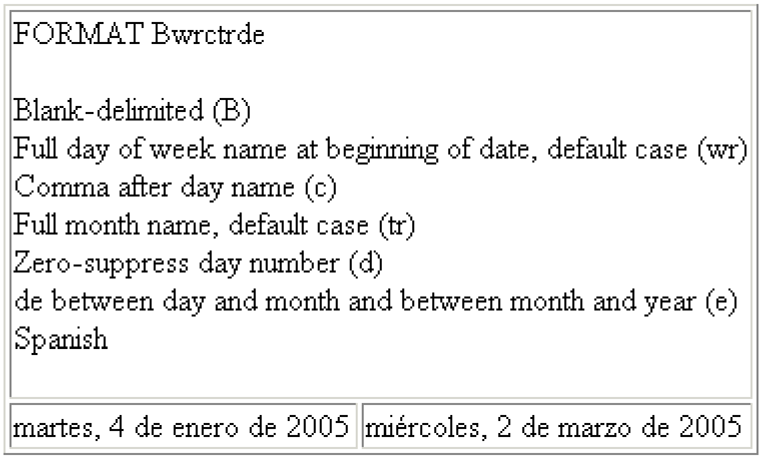
Blank-delimited (B)
Full day of week name at beginning of date, default case (wr)
Full month name, default case (tr)
English

mardi 04 janvier 2005	mercredi 02 mars 2005

The following request prints a blank delimited date in Spanish with a full day name at the beginning in lowercase (the default for Spanish) followed by a comma, and with the word de between the day number and month and between the month and year:

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1      ;
DATEW2/W=TRANS2     ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT5A/A30=DATETRAN(DATEYYMD,  '(DMYY)', '(Bwrctrde)', 'ES', 30, 'A30');
OUT5B/A30=DATETRAN(DATEYYMD2, '(DMYY)', '(Bwrctrde)', 'ES', 30, 'A30');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Bwrctrde"
""
"Blank-delimited (B)"
"Full day of week name at beginning of date, default case (wr)"
"Comma after day name (c)"
"Full month name, default case (tr)"
"Zero-suppress day number (d)"
"de between day and month and between month and year (e)"
"Spanish"
""
SUM OUT5A AS '' OUT5B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



FORMAT Bwrctrde

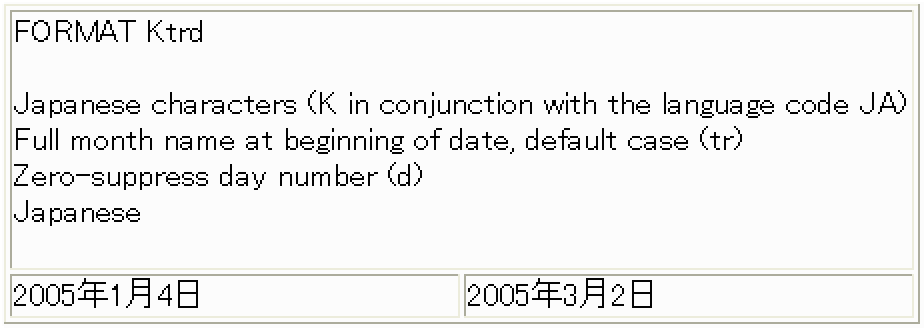
Blank-delimited (B)
Full day of week name at beginning of date, default case (wr)
Comma after day name (c)
Full month name, default case (tr)
Zero-suppress day number (d)
de between day and month and between month and year (e)
Spanish

martes, 4 de enero de 2005	miércoles, 2 de marzo de 2005

The following request prints a date in Japanese characters with a full month name at the beginning, in the default case and with zero suppression:

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1    ;
DATEYYMD2/YYMDW=TRANS2   ;

OUT6A/A30=DATETRAN(DATEYYMD , '(YYMD)', '(Ktrd)', 'JA', 30, 'A30');
OUT6B/A30=DATETRAN(DATEYYMD2, '(YYMD)', '(Ktrd)', 'JA', 30, 'A30');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Ktrd"
""
"Japanese characters (K in conjunction with the language code JA)"
"Full month name at beginning of date, default case (tr)"
"Zero-suppress day number (d)"
"Japanese"
""
SUM OUT6A AS '' OUT6B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



FORMAT Ktrd

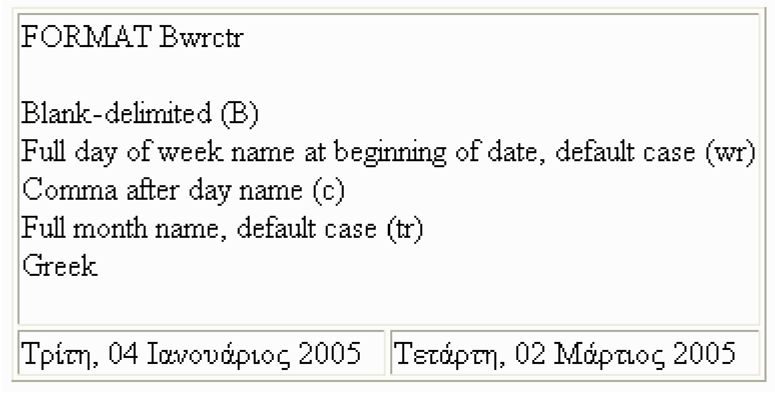
Japanese characters (K in conjunction with the language code JA)
Full month name at beginning of date, default case (tr)
Zero-suppress day number (d)
Japanese

2005年1月4日	2005年3月2日

The following request prints a blank delimited date in Greek with a full day name at the beginning in the default case followed by a comma, and with a full month name in the default case:

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1      ;
DATEW2/W=TRANS2     ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT7A/A30=DATETRAN(DATEYYMD , '(DMYY)', '(Bwrctr)', 'GR', 30, 'A30');
OUT7B/A30=DATETRAN(DATEYYMD2, '(DMYY)', '(Bwrctr)', 'GR', 30, 'A30');
END
TABLE FILE VIDEOTRK
HEADING
"FORMAT Bwrctrde"
""
"Blank-delimited (B)"
"Full day of week name at beginning of date, default case (wr)"
"Comma after day name (c)"
"Full month name, default case (tr)"
"Greek"
""
SUM OUT7A AS '' OUT7B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



FORMAT Bwrctr

Blank-delimited (B)
Full day of week name at beginning of date, default case (wr)
Comma after day name (c)
Full month name, default case (tr)
Greek

Τρίτη, 04 Ιανουάριος 2005	Τετάρτη, 02 Μάρτιος 2005

Extensions to Date-Time Formats

How to:

Enable ISO Standard Date-Time Notation

Specify Universal Time Format

Specify the Character T as the Date-Time Separator

Reference:

Abbreviations for Component Name Argument for Date-Time Functions

Example:

Specifying Date-Time Input Values

Specifying an Abbreviated Component Name in a Date-Time Function

Date-time formats can produce output values and accept input values that are compatible with the ISO 8601:2000 date-time notation standard. A SET parameter and additional formatting options have been added to enable this notation. In addition, abbreviations are supported for date-time components used as arguments to date-time functions.

Syntax: How to Enable ISO Standard Date-Time Notation

```
SET DTSTANDARD = {OFF|ON|STANDARD|STANDARDU}
```

where:

`OFF`

Does not provide compatibility with the ISO 8601:2000 date-time notation standard. OFF is the default value.

`ON|STANDARD`

Enables recognition and output of the ISO standard formats, including use of T as the delimiter between date and time, use of period or comma as the delimiter of fractional seconds, use of Z at the end of “universal” times, and acceptance of inputs with time zone information. STANDARD is a synonym for ON.

`STANDARDU`

Enables ISO standard formats (like STANDARD) and also, where possible, converts input strings to the equivalent "universal" time (formerly known as "Greenwich Mean Time"), thus enabling applications to store all date-time values in a consistent way.

Syntax: How to Specify Universal Time Format

```
HtimeZ
```

where:

`time`

Is a format option for the time components. For example HHI specifies a time component consisting of a two-digit hour and a two-digit minute.

`Z`

Indicates universal time. Z is incompatible with AM/PM output. It prints Z at the end of the ouput string to indicate a universal time.

Syntax: How to Specify the Character T as the Date-Time Separator

With the STANDARD and STANDARDU settings, the separator for dates is always a hyphen.

The separator between date and time is blank by default. However, if you specify the following separator option, the date and time are separated by the character T:

`HdateUtime[Z]`

where:

date

Is a format option for the date components. For example HYYMD specifies a date component consisting of a four-digit year, a two-digit month, and a two-digit day.

`U`

Is the date separator option that displays the character T between the date and time. U is incompatible with AM/PM output.

time

Is a format option for the time components. For example, I specifies a time component consisting of a two-digit hour and a two-digit minute.

`Z`

Indicates universal time.

Example: Specifying Date-Time Input Values

With DTSTANDARD settings of STANDARD and STANDARDU, the following time formats can be read as input:

Input Value	Description
`14:30[:20,99]`	Comma separates time components instead of period.
`14:30[:20.99]Z`	Universal time.

`15:30[:20,99]+01`
`15:30[:20,99]+0100`
`15:30[:20,99]+01:00`
	Each of these is the same as above in Central European Time.

`09:30[:20.99]-05`	Same as above in Eastern Standard Time.

Note that these values are stored identically internally with the STANDARDU setting. With the STANDARD setting, everything following the Z, +, or - is ignored.

Reference: Abbreviations for Component Name Argument for Date-Time Functions

The following component names and abbreviations are supported when calling a date-time function. Note that the component names and abbreviations can be entered in uppercase or lowercase:

Component Name	**Abbreviation**	**Values**
year	yy	0001-9999
quarter	qq	1-4
month	mm	1-12
day-of-year	dy	1-366
day or day-of-month	dd	1-31
week	wk	1-53
weekday	dw	1-7 (Sunday-Saturday)
hour	hh	0-23
minute	mi	0-59
second	ss	0-59
millisecond	ms	0-999
microsecond	mc	0-999999

Example: Specifying an Abbreviated Component Name in a Date-Time Function

The following request finds the number of days between the ADD_MONTH and TRANSDATE fields. The month component in the call to the HADD function is abbreviated as mm. The day component in the call to the HDIFF function is abbreviated as DD.

Standard universal date output is specified with the Z option in the ADD_MONTH format. The U separator option displays the date and time separated by the character T:

```
SET DTSTANDARD = STANDARD
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDUSZ = HADD(TRANSDATE, 'mm', 2, 8, 'HYYMDUSZ');
DIFF_DAYS/D12.2 = HDIFF(ADD_MONTH, TRANSDATE, 'DD', 'D12.2');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         ADD_MONTH                  DIFF_DAYS
------  ---------         ---------                  ---------
1118    2000/06/26 05:45  2000-08-26T05:45:00Z           61.00
1237    2000/02/05 03:30  2000-04-05T03:30:00Z           60.00
```

MIRR Function

How to:

Calculate the Modified Internal Rate of Return

Reference:

Usage Notes for the MIRR Function

Example:

Calculating the Modified Internal Rate of Return

The MIRR function calculates the modified internal rate of return for a series of periodic cash flows.

Syntax: How to Calculate the Modified Internal Rate of Return

```
TABLE FILE ...
{PRINT|SUM} field ...
COMPUTE rrate/fmt = MIRR(cashflow, finrate, reinvrate, output);
WITHIN {sort_field|TABLE}
```

where:

field ...

Are fields that appear in the report output.

rrate

Is the field that contains the calculated return rate.

fmt

Is the format of the return rate. The data type must be D.

cashflow

Is a numeric field. Each value represents either a payment (negative value) or income (positive value) for one period. The values must be in the correct sequence in order for for the sequence of cash flows to be calculated correctly. The dates corresponding to each cash flow should be equally spaced and sorted in chronological order. The calculation requires at least one negative value and one positive value in the *cashflow* field. If the values are all positive or all negative, a zero result is returned.

finrate

Is a finance rate for negative cash flows. This value must be expressed as a non-negative decimal fraction between 0 and 1. It must be constant within each sort group for which a return rate is calculated, but it can change between sort groups.

reinvrate

Is the reinvestment rate for positive cash flows. This value must be expressed as a non-negative decimal fraction between 0 and 1. It must be constant within each sort group but can change between sort groups. It must be constant within each sort group for which a return rate is calculated, but it can change between sort groups.

output

Is the name of the field that contains the return rate, or its format enclosed in single quotation marks.

sort_field

Is a field that sorts the report output and groups it into subsets of rows on which the function can be calculated separately. To calculate the function using every row of the report output, use the WITHIN TABLE phrase. A WITHIN phrase is required.

Reference: Usage Notes for the MIRR Function

- ❏ This function is only supported in a COMPUTE command with the WITHIN phrase.
- ❏ The cash flow field must contain at least one negative value and one positive value.
- ❏ Dates must be equally spaced.
- ❏ Missing cash flows or dates are not supported.

Example: Calculating the Modified Internal Rate of Return

The following request calculates modified internal return rates for categories of products. It assumes a finance charge of ten percent and a reinvestment rate of ten percent. The request is sorted by date so that the correct cash flows are calculated. The rate returned by the function is multiplied by 100 in order to express it as a percent rather than a decimal value. Note that the format includes the % character. This causes a percent symbol to display, but it does not calculate a percent.

In order to create one cash flow value per date, the values are summed. NEWDOLL is defined in order to create negative values in each category as required by the function:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 NEWDOLL/D12.2 = IF DATE LT '19970401' THEN -1 * DOLLARS ELSE DOLLARS;
END

TABLE FILE GGSALES
  SUM NEWDOLL
  COMPUTE RRATE/D7.2% = MIRR(NEWDOLL, .1, .1, RRATE) * 100;
  WITHIN CATEGORY
  BY CATEGORY
  BY SDATE
  WHERE SYEAR EQ 97
END
```

A separate rate is calculated for each category because of the WITHIN CATEGORY phrase. A portion of the output is shown:

```
Category      SDATE            NEWDOLL      RRATE
--------      -----            -------      -----
Coffee        1997/01      -801,123.00     15.11%
              1997/02      -682,340.00     15.11%
              1997/03      -765,078.00     15.11%
              1997/04       691,274.00     15.11%
              1997/05       720,444.00     15.11%
              1997/06       742,457.00     15.11%
              1997/07       747,253.00     15.11%
              1997/08       655,896.00     15.11%
              1997/09       730,317.00     15.11%
              1997/10       724,412.00     15.11%
              1997/11       620,264.00     15.11%
              1997/12       762,328.00     15.11%
Food          1997/01      -672,727.00     16.24%
              1997/02      -699,073.00     16.24%
              1997/03      -642,802.00     16.24%
              1997/04       718,514.00     16.24%
              1997/05       660,740.00     16.24%
              1997/06       734,705.00     16.24%
              1997/07       760,586.00     16.24%
```

To calculate one modified internal return rate for all of the report data, use the WITHIN TABLE phrase. In this case, the data does not have to be sorted by CATEGORY:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 NEWDOLL/D12.2 = IF DATE LT '19970401' THEN -1 * DOLLARS ELSE DOLLARS;
END

TABLE FILE GGSALES
  SUM NEWDOLL
  COMPUTE RRATE/D7.2% = MIRR(NEWDOLL, .1, .1, RRATE) * 100;
  WITHIN TABLE
  BY SDATE
  WHERE SYEAR EQ 97
END
```

The output is:

```
SDATE            NEWDOLL      RRATE
-----            -------      -----
1997/01    -1,864,129.00     15.92%
1997/02    -1,861,639.00     15.92%
1997/03    -1,874,439.00     15.92%
1997/04     1,829,838.00     15.92%
1997/05     1,899,494.00     15.92%
1997/06     1,932,630.00     15.92%
1997/07     2,005,402.00     15.92%
1997/08     1,838,863.00     15.92%
1997/09     1,893,944.00     15.92%
1997/10     1,933,705.00     15.92%
1997/11     1,865,982.00     15.92%
1997/12     2,053,923.00     15.92%
```

XIRR Function

How to:

Calculate the Internal Rate of Return

Reference:

Usage Notes for the XIRR Function

Example:

Calculating the Internal Rate of Return

The XIRR function calculates the internal rate of return for a series of cash flows that can be periodic or non-periodic.

Syntax: How to Calculate the Internal Rate of Return

```
TABLE FILE ...
{PRINT|SUM} field ...
COMPUTE rrate/fmt = XIRR (cashflow, dates, guess, maxiterations, output);
WITHIN {sort_field|TABLE}
```

where:

field ...

Are fields that appear in the report output.

rrate

Is the field that contains the calculated return rate.

`fmt`

Is the format of the return rate. The data type must be D.

`cashflow`

Is a numeric field. Each value of this field represents either a payment (negative value) or income (positive value) for one period. The values must be in the correct sequence in order for the sequence of cash flows to be calculated correctly. The dates corresponding to each cash flow should be equally spaced and sorted in chronological order. The calculation requires at least one negative value and one positive value in the *cashflow* field. If the values are all positive or all negative, a zero result is returned.

`dates`

Is a date field containing the cash flow dates. The dates must be full component dates with year, month, and day components. Dates cannot be stored in fields with format A, I, or P. They must be stored in date fields (for example, format YMD, not AYMD). There must be the same number of dates as there are cash flow values. The number of dates must be the same as the number of cash flows.

`guess`

Is an (optional) initial estimate of the expected return rate expressed as a decimal. The default value is .1 (10%). To accept the default, supply the value 0 (zero) for this argument.

`maxiterations`

Is an (optional) number specifying the maximum number of iterations that can be used to resolve the rate using Newton's method. 50 is the default value. To accept the default, supply the value 0 (zero) for this argument. The rate is considered to be resolved when successive iterations do not differ by more than 0.0000003. If this level of accuracy is achieved within the maximum number of iterations, calculation stops at that point. If it is not achieved after reaching the maximum number of iterations, calculation stops and the value calculated by the last iteration is returned.

`output`

D

Is the name of the field that contains the return rate, or its format enclosed in single quotation marks.

`sort_field`

Is a field that sorts the report output and groups it into subsets of rows on which the function can be calculated separately. To calculate the function using every row of the report output, use the WITHIN TABLE phrase. A WITHIN phrase is required.

Reference: Usage Notes for the XIRR Function

- This function is only supported in a COMPUTE command with the WITHIN phrase.
- The cash flow field must contain at least one negative value and one positive value.
- Dates cannot be stored in fields with format A, I, or P. They must be stored in date fields (for example, format YMD, not AYMD).
- Cash flows or dates with missing values are not supported.

Example: Calculating the Internal Rate of Return

The following request creates a FOCUS data source with cash flows and dates and calculates the internal return rate.

The Master File for the data source is:

```
FILENAME=XIRR01,SUFFIX=FOC
SEGNAME=SEG1,SEGTYPE=S1
FIELDNAME=DUMMY,FORMAT=A2,$
FIELDNAME=DATES,FORMAT=YYMD,$
FIELDNAME=CASHFL,FORMAT=D12.4,$
END
```

The procedure to create the data source is:

```
CREATE FILE XIRR01
MODIFY FILE XIRR01
FREEFORM DUMMY DATES CASHFL
DATA
AA,19980101,-10000. ,$
BB,19980301,2750.   ,$
CC,19981030,4250.   ,$
DD,19990215,3250.   ,$
EE,19990401,2750.   ,$
END
```

The request is sorted by date so that the correct cash flows can be calculated. The rate returned by the function is multiplied by 100 in order to express it as a percent rather than a decimal value. Note that the format includes the % character. This causes a percent symbol to display, but it does not calculate a percent:

```
TABLE FILE XIRR01
PRINT CASHFL
COMPUTE RATEX/D12.2%=XIRR(CASHFL, DATES, 0., 0., RATEX) * 100;
WITHIN TABLE
BY DATES
END
```

One rate is calculated for the entire report because of the WITHIN TABLE phrase:

```
DATES                 CASHFL             RATEX
-----                 ------             -----
1998/01/01    -10,000.0000              37.49%
1998/03/01      2,750.0000              37.49%
1998/10/30      4,250.0000              37.49%
1999/02/15      3,250.0000              37.49%
1999/04/01      2,750.0000              37.49%
```

PTOA Function

How to:

Convert a Packed Decimal Number to Alphanumeric Format

Example:

Converting From Packed to Alphanumeric Format

The PTOA function converts a packed decimal number from numeric format to alphanumeric format. It retains the decimal positions of the number and right-justifies it with leading spaces. You can also add edit options to a number converted by PTOA.

When using PTOA to convert a number containing decimals to a character string, you must specify an alphanumeric format large enough to accommodate both the integer and decimal portions of the number. For example, a P12.2C format is converted to A14. If the output format is not large enough, the rightmost characters are truncated.

Syntax: How to Convert a Packed Decimal Number to Alphanumeric Format

```
PTOA(number, '(format)', outfield)
```

where:

number

Numeric P (packed decimal)

Is the number to be converted, or the name of the field that contains the number.

format

Alphanumeric

Is the output format of the number enclosed in both single quotation marks and parentheses. Only packed decimal format is supported. Include any edit options that you want to display in the output.

The format value does not have to have the same length or number of decimal places as the original field. If you change the number of decimal places, the result is rounded. If you make the length too short to hold the integer portion of the number, asterisks display instead of the number.

If you use a field name for this argument, specify the name without quotation marks or parentheses. However, parentheses must be included around the format stored in this field. For example:

```
FMT/A10 = '(P12.2C)';
```

You can then use this field as the format argument when using the function in your request:

```
COMPUTE ALPHA_GROSS/A20 = PTOA(PGROSS, FMT, ALPHA_GROSS);
```

outfield

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The length of this argument must be greater than the length of *number* and must account for edit options and a possible negative sign.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

Example: Converting From Packed to Alphanumeric Format

PTOA is called twice to convert the PGROSS field from packed decimal to alphanumeric format. The format specified in the first call to the function is stored in a virtual field named FMT. The format specified in the second call to the function does not include decimal places, so the value is rounded when displayed:

```
DEFINE FILE EMPLOYEE
PGROSS/P18.2=GROSS;
FMT/A10='(P14.2C)';
END
TABLE FILE EMPLOYEE PRINT PGROSS NOPRINT
COMPUTE AGROSS/A17 = PTOA(PGROSS, FMT, AGROSS); AS ''
COMPUTE BGROSS/A37 = '<- THIS AMOUNT IS' |
                     PTOA(PGROSS, '(P5C)', 'A6') |
                     ' WHEN ROUNDED'; AS '' IN +1
BY HIGHEST 1 PAY_DATE NOPRINT
BY LAST_NAME NOPRINT
END
```

The output is:

```
2,475.00 <- THIS AMOUNT IS 2,475 WHEN ROUNDED
1,815.00 <- THIS AMOUNT IS 1,815 WHEN ROUNDED
2,255.00 <- THIS AMOUNT IS 2,255 WHEN ROUNDED
  750.00 <- THIS AMOUNT IS   750 WHEN ROUNDED
2,238.50 <- THIS AMOUNT IS 2,239 WHEN ROUNDED
1,540.00 <- THIS AMOUNT IS 1,540 WHEN ROUNDED
1,540.00 <- THIS AMOUNT IS 1,540 WHEN ROUNDED
1,342.00 <- THIS AMOUNT IS 1,342 WHEN ROUNDED
1,760.00 <- THIS AMOUNT IS 1,760 WHEN ROUNDED
1,100.00 <- THIS AMOUNT IS 1,100 WHEN ROUNDED
  791.67 <- THIS AMOUNT IS   792 WHEN ROUNDED
  916.67 <- THIS AMOUNT IS   917 WHEN ROUNDED
```

PUTDDREC and CLSDDREC Functions

How to:

Write a Character String as a Record in a Sequential File

Close All Files Opened by the PUTDDREC Function

Reference:

Usage Notes for the PUTDDREC and CLSDDREC Functions

Example:

Calling PUTDDREC in a TABLE Request

Calling PUTDDREC and CLSDDREC in Dialogue Manager -SET Commands

The PUTDDREC function writes a character string as a record in a flat file. The file must be identified with a CMS FILEDEF command (DYNAM or ALLOCATE on z/OS). If the file is defined with the APPEND option, the new record is appended. Without the APPEND option, the new record overwrites any existing file. For information about the FILEDEF command, see the *Overview and Operating Environments* manual.

If the file is not already open, PUTDDREC opens it the first time it is called. Each call to PUTDDREC can use the same file or open a new one. All of the files opened by PUTDDREC remain open until the end of a request or session. At the end of the request or session, all files opened by PUTDDREC are closed automatically.

If PUTDDREC is called in a Dialogue Manager -SET command, the files opened by PUTDDREC are not closed automatically until the end of a request or session. In this case, you can manually close the files and free the memory used to store information about open files by calling the CLSDDREC function.

Syntax: How to Write a Character String as a Record in a Sequential File

```
PUTDDREC(ddname, dd_len, record_string, record_len, outfield)
```

where:

ddname

Alphanumeric

Is the logical name assigned to the sequential file in a CMS FILEDEF command. For information about the CMS FILEDEF command, see the *Overview and Operating Environments* manual.

dd_len

Numeric

Is the number of characters in the logical name.

record_string

Alphanumeric

Is the character string to be added as the new record in the sequential file.

record_len

Numeric

Is the number of characters to add as the new record. It cannot be larger than the number of characters in *record_string*. To write all of *record_string* to the file, *record_len* should equal the number of characters in *record_string* and should not exceed the record length declared in the CMS FILEDEF command. If *record_len* is shorter than the length declared in the CMS FILEDEF command, the resulting file may contain extraneous characters at the end of each record. If *record_string* is longer than the length declared in the CMS FILEDEF command, *record_string* may be truncated in the resulting file.

outfield

Integer

Is the return code, which can have one of the following values:

Return Code	Description
0	Record is added.
-1	FILEDEF statement is not found.
-2	Error while opening the file.
-3	Error while adding the record to the file.

Syntax: How to Close All Files Opened by the PUTDDREC Function

```
CLSDDREC(outfield)
```

where:

outfield

Integer

Is the return code, which can be one of the following values:

Return Code	Description
0	Files are closed.
-1	Error while closing the files.

Reference: Usage Notes for the PUTDDREC and CLSDDREC Functions

- The open, close, and write operations are handled by the operating system. Therefore, the requirements for writing to the file and the results of deviating from the instructions when calling PUTDDREC are specific to your operating environment. Make sure you are familiar with and follow the guidelines for your operating system when performing input/output operations.
- You can call PUTDDREC in a DEFINE FILE command or in a DEFINE in the Master File. However, PUTDDREC does not open the file until its field name is referenced in a request.

Example: Calling PUTDDREC in a TABLE Request

The following example defines a new file whose logical name is PUTDD1. The TABLE request then calls PUTDDREC for each employee in the EMPLOYEE data source and writes a record to the file composed of the employee's last name, first name, employee ID, current job code, and current salary (converted to alphanumeric using the EDIT function). The return code of zero (in OUT1) indicates that the calls to PUTDDREC were successful:

```
CMS FILEDEF PUTDD1 DISK  PUTDD1 DATA A
-RUN
TABLE FILE EMPLOYEE
PRINT EMP_ID CURR_JOBCODE AS 'JOB' CURR_SAL
COMPUTE SALA/A12 = EDIT(CURR_SAL); NOPRINT
COMPUTE EMP1/A50= LAST_NAME|FIRST_NAME|EMP_ID|CURR_JOBCODE|SALA;
NOPRINT
COMPUTE OUT1/I1 = PUTDDREC('PUTDD1',6, EMP1, 50, OUT1);
BY LAST_NAME BY FIRST_NAME
END
```

The output is:

```
LAST_NAME        FIRST_NAME  EMP_ID     JOB          CURR_SAL  OUT1
---------        ----------  ------     ---          --------  ----
BANNING          JOHN        119329144  A17        $29,700.00     0
BLACKWOOD        ROSEMARIE   326179357  B04        $21,780.00     0
CROSS            BARBARA     818692173  A17        $27,062.00     0
GREENSPAN        MARY        543729165  A07         $9,000.00     0
IRVING           JOAN        123764317  A15        $26,862.00     0
JONES            DIANE       117593129  B03        $18,480.00     0
MCCOY            JOHN        219984371  B02        $18,480.00     0
MCKNIGHT         ROGER       451123478  B02        $16,100.00     0
ROMANS           ANTHONY     126724188  B04        $21,120.00     0
SMITH            MARY        112847612  B14        $13,200.00     0
                 RICHARD     119265415  A01         $9,500.00     0
STEVENS          ALFRED      071382660  A07        $11,000.00     0
```

After running this request, the sequential file contains the following records:

```
BANNING         JOHN      119329144A17000000029700
BLACKWOOD       ROSEMARIE 326179357B04000000021780
CROSS           BARBARA   818692173A17000000027062
GREENSPAN       MARY      543729165A07000000009000
IRVING          JOAN      123764317A15000000026862
JONES           DIANE     117593129B03000000018480
MCCOY           JOHN      219984371B02000000018480
MCKNIGHT        ROGER     451123478B02000000016100
ROMANS          ANTHONY   126724188B04000000021120
SMITH           MARY      112847612B14000000013200
SMITH           RICHARD   119265415A01000000009500
STEVENS         ALFRED    071382660A07000000011000
```

Example: Calling PUTDDREC and CLSDDREC in Dialogue Manager -SET Commands

The following example defines a new file whose logical name is PUTDD1. The first -SET command creates a record to add to this file. The second -SET command calls PUTDDREC to add the record. The last -SET command calls CLSDDREC to close the file. The return codes are displayed to make sure operations were successful:

```
CMS FILEDEF PUTDD1 DISK  PUTDD1 DATA A
-RUN
-SET &EMP1 = 'SMITH'|'MARY'|'A07'|'27000';
-TYPE DATA = &EMP1
-SET &OUT1 = PUTDDREC('PUTDD1',6, &EMP1, 17, 'I1');
-TYPE PUT RESULT = &OUT1
-SET &OUT1 = CLSDDREC('I1');
-TYPE CLOSE RESULT = &OUT1
```

The output is:

```
DATA = SMITHMARYA0727000
PUT RESULT = 0
CLOSE RESULT = 0
```

After running this procedure, the sequential file contains the following record:

```
SMITHMARYA0727000
```

SET ERROROUT = OVERRIDE

In this section:

Default Error Handling with ERROROUT ON or OFF

How to:

Override Error or Warning Messages

Revert to Default Error Handling

Reference:

Usage Notes for SET ERROROUT=OVERRIDE/filename

Example:

Overriding Standard Error Messages

The ERROROUT=OVERRIDE/*filename* setting enables you to change the severity of error conditions encountered in FOCUS processing. You can change errors to warnings or warnings to errors for specified error messages or ranges of error messages.

Note: The ERROROUT setting is ignored in interactive sessions.

Default Error Handling with ERROROUT ON or OFF

With ERROROUT=ON, any error message terminates the job step and results in a return code of 8. For warning messages, you can test a Dialogue Manager variable (&FOCERRNUM) to determine whether or not you wish to continue based on the actual return code. ERROROUT=ON is only supported in batch jobs.

With ERROROUT=OFF, the application is *permitted* to continue regardless of the condition's severity level, moving ahead to the next control point within the application. OFF is the default value.

As with ERROROUT=ON, warning messages do not invoke this behavior and all actions are left to the user.

By creating an override file, you can redefine the severity of error or warning conditions encountered, causing an application to terminate only under conditions that you consider serious in the context of your application environment. You might, for example, determine that certain default error conditions that normally terminate processing are not serious in your current context, and therefore merit only warnings. With the override file, the application processes as if the ERROROUT setting is OFF but with the changes in severity level indicated by the override file.

Procedure: How to Change Message Severity Levels

To override message severity levels:

1. Create a message severity override file. The file must be a sequential file sorted in order of message number. The format of each line is described in *How to Override Error or Warning Messages*.

2. Identify the message severity override file by including the following command in a profile or FOCEXEC.

   ```
   SET ERROROUT=[OVERRIDE/]filename
   ```

   where:

   filename

   Is the name of the override file.On z/OS, *filename* must be a member in the concatenation of data sets allocated to DDNAME ERRORS. On z/VM, *filename* is the name of the override file. The override file must have filetype ERRORS.

Syntax: How to Override Error or Warning Messages

To implement an override file, use the following syntax. Note that you cannot set this parameter with an ON TABLE SET command.

```
SET ERROROUT=[OVERRIDE/]filename
```

where:

`filename`

Is the name of the file containing override entries. OVERRIDE/*filename* is a synonym for *filename*. The file must be sorted in ascending order of message number. Each entry has the following form:

```
message_number {Warning|W|Error|E}
```

or

```
start_message_number - end_message_number {Warning|W|Error|E}
```

where:

`message_number` is the number of a message whose severity level you want to redefine.

`Warning` or `W` defines the message as a warning message. The option is not case-sensitive.

`Error` or `E` defines the message as an error message. The option is not case-sensitive.

`start_message_number` is the beginning of a range of message numbers that will be assigned the same severity level.

`end_message_number` is the end of the range of message numbers.

Any combination of single messages and message ranges may be included in the override file. Each item must be separated from the next item by a space.

Syntax: How to Revert to Default Error Handling

To revert to default error-condition handling after using a message severity override file, issue the following command:

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

where:

`ON`

Reinstates the default message file.

The application terminates if a condition is an error, and can continue if a condition is a warning.

`OFF`

Reinstates the default message file. OFF is the default value.

With ERROROUT=OFF, the application is permitted to continue to the next control point within the application regardless of the severity level of the error posted.

Example: Overriding Standard Error Messages

Consider the following procedure that generates two warning messages (FOC095 and FOC096) about the SKIP-LINE and SUBFOOT commands:

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

With ERROROUT=OFF, the warning messages display and then the report is generated:

```
 ERROR AT OR NEAR LINE     14  IN PROCEDURE OVERRIDEFOCEXEC *
(FOC095) WARNING. PREVIOUS USE OF THIS OPTION IS OVERRIDDEN: SKIP-LINE
 ERROR AT OR NEAR LINE     17  IN PROCEDURE OVERRIDEFOCEXEC *
(FOC096) WARNING. NO TEXT SUPPLIED BELOW SUBHEAD OR SUBFOOT
 ERROR AT OR NEAR LINE     17  IN PROCEDURE OVERRIDEFOCEXEC *
(FOC096) WARNING. NO TEXT SUPPLIED BELOW SUBHEAD OR SUBFOOT
 NUMBER OF RECORDS IN TABLE=       12  LINES=     12

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
             Departmental Salary Report

                                     Current
Department  Last Name                Salary   Job Code
----------  ---------                -------  --------

MIS         BLACKWOOD             $21,780.00  B04
            CROSS                 $27,062.00  A17
            *** WARNING: CROSS 's salary exceeds recommended guidelines.
            GREENSPAN              $9,000.00  A07
            JONES                 $18,480.00  B03
            MCCOY                 $18,480.00  B02
            SMITH                 $13,200.00  B14

PRODUCTION  BANNING               $29,700.00  A17
           *** WARNING: BANNING 's salary exceeds recommended guidelines.
            IRVING                $26,862.00  A15
            MCKNIGHT              $16,100.00  B02
            ROMANS                $21,120.00  B04
            SMITH                  $9,500.00  A01
            STEVENS               $11,000.00  A07
```

The ERRORS file named NEWERR identifies 095 and 096 as errors rather than warnings:

```
095 E
096 E
```

Running the same request with the following setting causes the application to terminate without displaying the report output:

```
SET ERROROUT = NEWERR
```

The following messages display with this setting in effect:

```
 ERROR AT OR NEAR LINE     14  IN PROCEDURE OVERRIDEFOCEXEC *
(FOC095) WARNING. PREVIOUS USE OF THIS OPTION IS OVERRIDDEN: SKIP-LINE
Exiting due to Exit on Error...
 ERROR AT OR NEAR LINE     17  IN PROCEDURE OVERRIDEFOCEXEC *
(FOC096) WARNING. NO TEXT SUPPLIED BELOW SUBHEAD OR SUBFOOT
Exiting due to Exit on Error...
 ERROR AT OR NEAR LINE     17  IN PROCEDURE OVERRIDEFOCEXEC *
(FOC096) WARNING. NO TEXT SUPPLIED BELOW SUBHEAD OR SUBFOOT
Exiting due to Exit on Error...
 ...RETRIEVAL KILLED
(FOC026) THE REPORT IS NO LONGER AVAILABLE
Exiting due to Exit on Error...
```

Reference: Usage Notes for SET ERROROUT=OVERRIDE/filename

For additional details about error handling, see your documentation for SET ERROROUT=ON or OFF. In addition:

- You cannot set the ERROROUT=*filename* parameter in an ON TABLE SET command.
- ERROROUT settings apply only to batch operations.

STRREP Function

How to:

Replace Character Strings

Reference:

Usage Notes for STRREP Function

Example:

Replacing Commas and Dollar Signs

The STRREP function enables you to replace all instances of a specified string within a given input string. It also supports replacement by null strings.

Syntax: How to Replace Character Strings

`STRREP (inlength, instring, searchlength, searchstring, replength, repstring, outlength, outstring)`

where:

inlength

Numeric

Is the number of characters in the input string.

instring

Alphanumeric

Is the input string.

searchlength

Numeric

Is the number of characters in the (shorter length) string to be replaced.

searchstring

Alphanumeric

Is the character string to be replaced.

replength

Numeric

Is the number of characters in the replacement string. Must be zero (0) or greater. Zero is used to delete a character from a string.

repstring

Alphanumeric

Is the replacement string (alphanumeric). Ignored if replength is zero (0).

outlength

Numeric

Is the number of characters in the resulting output string. Must be 1 or greater.

outstring

Alphanumeric

Is the resulting output string after all replacements and padding.

Reference: Usage Notes for STRREP Function

The maximum string length is 4095.

Example: Replacing Commas and Dollar Signs

In the following example, STRREP finds and replaces commas and dollar signs that appear in the CS_ALPHA field, first replacing commas with null strings to produce CS_NOCOMMAS (removing the commas) and then replacing the dollar signs ($) with (USD) in the right-most CURR_SAL column:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL NOPRINT
COMPUTE CS_ALPHA/A15=FTOA(CURR_SAL,'(D12.2M)',CS_ALPHA);
        CS_NOCOMMAS/A14=STRREP(15,CS_ALPHA,1,',',0,'X',14,CS_NOCOMMAS);
        CS_USD/A17=STRREP(14,CS_NOCOMMAS,1,'$',4,'USD ',17,CS_USD);
        NOPRINT
        CS_USD/R AS CURR_SAL
BY LAST_NAME
END
```

The output is:

```
LAST_NAME         CS_ALPHA          CS_NOCOMMAS              CURR_SAL
---------         --------          -----------      -----------------
BANNING                $29,700.00        $29700.00        USD 29700.00
BLACKWOOD              $21,780.00        $21780.00        USD 21780.00
CROSS                  $27,062.00        $27062.00        USD 27062.00
GREENSPAN               $9,000.00         $9000.00         USD 9000.00
IRVING                 $26,862.00        $26862.00        USD 26862.00
JONES                  $18,480.00        $18480.00        USD 18480.00
MCCOY                  $18,480.00        $18480.00        USD 18480.00
MCKNIGHT               $16,100.00        $16100.00        USD 16100.00
ROMANS                 $21,120.00        $21120.00        USD 21120.00
SMITH                  $22,700.00        $22700.00        USD 22700.00
STEVENS                $11,000.00        $11000.00        USD 11000.00
```

FOCREPLAY: Recording and Playing Back a FOCUS Session

FOCREPLAY is a tool that enables you to record your keystrokes when executing an interactive FOCUS application and play them back at some time in the future. You invoke the FOCREPLAY functions by allocating the DDNAMEs FOCREPLAY uses for recording and playing back a FOCUS session.

The recording function of FOCREPLAY is called *input mode*. In this mode, you are in charge of the keyboard. It produces two outputs:

❏ A script that you can later use to replay the application.

❏ An output file containing the captured FOCUS session, including any screens displayed during the session.

To play back the script you invoke *replay mode*, which is activated by allocating a previously recorded script as the input to FOCREPLAY. Replay mode can be run in batch or interactively. During an interactive replay session, FOCREPLAY is in charge of the keyboard. It presents each line in the script and waits for a response from you in order to proceed. Replay mode produces an output file containing the replayed session.

You can then use any file comparison tool installed at your site to compare the original session's output to the replayed session's output.

One of the uses for this tool is to simplify acceptance testing of a new release of FOCUS. If you record an application's execution in your current release and then upgrade to a new release, you can play back the script and compare the resulting output file with the results of the original run to determine if the application produces the same output under the new release.

Reference: Usage Notes and Prerequisites for FOCREPLAY

- FOCREPLAY was tested with IBM's PC3270 emulator. However, it should work with all 3270 emulators such as EXTRA, Reflections, and Hummingbird if they use standard 3270 job stream calls.
- In order to judge the results of the comparison between the original output and the replay, you must understand any factors in the environment may be different and may affect the outcome. Some examples are expired fields, dates that have passed and are no longer valid, changes to any screen or menu, or the date and time of running the application.
- You should carefully document the conditions for running the script so that you can replicate these conditions accurately when replaying it.

  For example, assume your application contains a MODIFY or Maintain request that includes, updates, or deletes instances from a data source. In order to replay the session and get the same results, you need to run it against a copy of the data source as it was prior to the original run, and you need to run it against the transactions used in the original run.

  You also must keep a record of the operating system release and service level, database engine release, and FOCUS release.
- FOCREPLAY does not do the comparison between the original run and the replay. You can use any file comparison tool you have installed at your site, for example XCOMPARE on z/VM or ISPF 3.13 on z/OS.
- FOCREPLAY cannot be used with any feature that requires allocations outside of FOCUS (for example, FOCCALC or HLI) or with any feature that starts a service (such as MSO). Output of operating system commands is not captured.

Reference: Troubleshooting FOCREPLAY Applications

Before calling in a problem, make sure you have accounted for acceptable differences in the comparison between two releases. For example, when you run your application the second time and you want to compare its output to that of your initial (base) execution of the application, dates, release levels, and operating system messages will be different from the base. Remove these acceptable differences from the comparison between the two runs.

Configuring FOCREPLAY

How to:

Activate FOCREPLAY Input Mode

Issue FILEDEFs for Input Mode on z/VM

Allocate DDNAMEs for Input Mode on z/OS

Activate FOCREPLAY Replay Mode

Issue FILEDEFs for Replay Mode on z/VM

Allocate DDNAMEs for Replay Mode on z/OS

Example:

Sample FILEDEFs for Input Mode on z/VM

Allocating DDNAMEs for Input Mode on z/OS

Sample FILEDEFs for Replay Mode on z/VM

Allocating DDNAMEs for Replay Mode on z/OS

FOCREPLAY is activated by allocating the DDNAMEs it uses to record, capture, and/or replay a FOCUS session. You can add these allocations to your REXX EXEC, CLIST, or batch JCL that invokes FOCUS.

The required disposition for the FOCREPLAY data sets is DISP=MOD. The recommended DCB attributes are RECFM=VB, LRECL=233472, BLKSIZE=23476 (half track blocking), DSORG=PS. If you want the data sets to be members of a library, you can allocate them as members of a PDSE. Because standard PDS data sets do not support DISP=MOD allocation for their members, you cannot allocate the data sets as members of a PDS. If you want to store them in a PDS, you can allocate them as sequential files and copy them to PDS members after they are complete.

If the DCB attributes differ from the recommended values, they will be changed dynamically during the session. You must have read/write access to all of the FOCREPLAY data sets.

Note:

- You can make limited edits to a recorded script file. For example, you can insert a break at any point if you want to execute the script up to that point and then stop. The TED editor cannot edit these files. You must use a system editor such as XEDIT on z/VM or the ISPF editor on z/OS to edit them. You can access the system editor from FOCUS using the IEDIT command.
- The FOCREPLAY output files are created in binary mode with required blanks at the end of some of the records. In the ISPF editor, you must issue the PRESERVE ON command to retain these blanks.

Procedure: How to Activate FOCREPLAY Input Mode

FOCREPLAY records your FOCUS session if you allocate the following two DDNAMEs:

- **FSC3270I.** This DDNAME will contain the script of the session. You can play back this script to rerun the session by allocating the script file as input to replay mode. The 3270 terminal characteristics and session mode (height, width, graphic support, colors, and external attributes) are automatically recorded in this file. This enables you to record sessions for different types of terminals or different modes. Do not edit these lines.
- **FSC3270O.** This DDNAME will contain the captured FOCUS session.

If you allocate an existing file to either DDNAME, the new script and session output are appended to the existing information in the files.

Syntax: How to Issue FILEDEFs for Input Mode on z/VM

```
FILEDEF FSC3270I DISK script_file ft1 fm (DISP MOD
FILEDEF FSC3270O DISK session_output ft2 fm (DISP MOD
```

where:

script_file ft1 fm

Is the filename, filetype, and filemode of the file that will contain the script of the FOCUS session.

session_output ft2 fm

Is the filename, filetype, and filemode of the file that will contain the captured FOCUS session.

Example: Sample FILEDEFs for Input Mode on z/VM

```
FILEDEF FSC3270I DISK FSC3270 IN A (DISP MOD
FILEDEF FSC3270O DISK FSC3270 OUT A (DISP MOD
```

Syntax: How to Allocate DDNAMEs for Input Mode on z/OS

```
ALLOC F(FSC3270I) DA('hlq.script.file') MOD
ALLOC F(FSC3270O) DA('hlq.session.output') MOD
```

where:

hlq.script.file

Is the data set that will contain the script of the FOCUS session.

hlq.session.output

Is the data set that will contain the captured FOCUS session.

Example: Allocating DDNAMEs for Input Mode on z/OS

```
ALLOC F(FSC3270I) DA('USER1.FSC3270I.DATA') MOD
ALLOC F(FSC3270O) DA('USER1.FSC3270O.DATA') MOD
```

Procedure: How to Activate FOCREPLAY Replay Mode

FOCREPLAY replays your FOCUS session and creates a new output file containing the replayed session if you allocate the following three DDNAMEs:

❑ **FSC3270Q.** This DDNAME must be allocated to the script of the previously recorded session that you want to play back.

❑ **FSC3270O.** This DDNAME will contain the replayed FOCUS session. If you allocate the same file that was used when you recorded the session, the played back session will be appended to the recorded session. If you allocate a different output file, the played back session will be in a separate file from the original session.

❑ **FSC3270I.** This DDNAME should be allocated as DUMMY when replaying a session. If you allocate it to a file, you will both play back the session and record a new input script in the allocated file. This is necessary to use the <RECORD> function described later.

Syntax: How to Issue FILEDEFs for Replay Mode on z/VM

```
FILEDEF FSC3270Q DISK script_file ft1 fm (DISP MOD
FILEDEF FSC3270O DISK session2_output ft2 fm (DISP MOD
FILEDEF FSC3270I DUMMY
```

where:

script_file ft1 fm

Is the filename, filetype, and filemode of the file that contains the script of the previously recorded FOCUS session.

session2_output ft2 fm

Is the filename, filetype, and filemode of the file that will contain the replayed session.

Example: Sample FILEDEFs for Replay Mode on z/VM

```
FILEDEF FSC3270Q DISK FSC3270 IN A (DISP MOD
FILEDEF FSC3270O DISK FSC3270 OUT2 A (DISP MOD
FILEDEF FSC3270I DUMMY
```

Syntax: How to Allocate DDNAMEs for Replay Mode on z/OS

In a CLIST:

```
ALLOC F(FSC3270Q) DA('hlq.script.file') MOD
ALLOC F(FSC3270O) DA('hlq.session2.output') MOD
ALLOC (FSC3270I) DUMMY
```

In a batch job:

```
//FSC3270Q DD DISP=MOD,DSN=hlq.script.file
//FSC3270O DD DISP=MOD,DSN=hlq.session2.output
//FSC3270I DD DUMMY
```

where:

hlq.script.file

Is the data set that contains the script of the previously recorded FOCUS session.

hlq.session2.output

Is the data set that will contain the replayed session.

Example: Allocating DDNAMEs for Replay Mode on z/OS

In a CLIST:

```
ALLOC F(FSC3270Q) DA('USER1.FSC3270I.DATA') MOD
ALLOC F(FSC3270O) DA('USER1.FSC3270O.DATA2') MOD
ALLOC F(FSC3270I) DUMMY
```

In a batch job:

```
//FSC3270Q DD DISP=MOD,DSN=USER1.FSC3270I.DATA
//FSC3270O DD DISP=MOD,DSN=USER1.FSC3270O.DATA2
//FSC3270I DD DUMMY
```

Recording a FOCUS Session

Example:

Recording a Session Under z/VM

Recording a Session Under z/OS

This section illustrates how to record a FOCUS session under z/VM and z/OS.

Example: Recording a Session Under z/VM

Issue FILEDEF commands for the FOCREPLAY input mode DDNAMEs prior to invoking FOCUS or in the REXX EXEC that invokes FOCUS. For example, the following FILEDEF commands define new data sets to contain the session script and the session output:

```
FILEDEF FSC3270I DISK FSSCRIPT IN A (DISP MOD
FILEDEF FSC3270O DISK FSC3270 OUTBASE A (DISP MOD
```

You must invoke FOCUS without a profile:

```
EX FOCUS (NOPROF
```

If you need a profile, execute the profile as the first command after invoking FOCUS.

The following FOCUS session issues a TABLE request against the MOVIES data source. This session is recorded by FOCREPLAY as a result of executing the REXX EXEC containing the FILEDEF commands for DDNAMEs FSC3270I and FSC3270O.

Issue the following TABLE request:

```
>  > TABLE FILE MOVIES
> PRINT RATING DIRECTOR TITLE/A20
> BY CATEGORY
> WHERE CATEGORY EQ 'DRAMA' OR 'MYSTERY' OR 'CHILDREN'
> END

 NUMBER OF RECORDS IN TABLE=       18  LINES=     18

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
```

Press Enter to display the first screen of the report:

```
PAGE     1


CATEGORY  RATING  DIRECTOR           TITLE
--------  ------  --------           -----
CHILDREN  NR                         SMURFS, THE
          G       BARTON C.          SHAGGY DOG, THE
          NR                         SCOOBY-DOO-A DOG IN
          G       GEROMINI           ALICE IN WONDERLAND
          NR                         SESAME STREET-BEDTIM
          NR                         ROMPER ROOM-ASK MISS
          NR      DISNEY W.          SLEEPING BEAUTY
          G       DISNEY W.          BAMBI
DRAMA     R       LUMET S.           DOG DAY AFTERNOON
MYSTERY   PG      HITCHCOCK A.       REAR WINDOW
          PG      HITCHCOCK A.       VERTIGO
          R       GRANT M.           FATAL ATTRACTION
          NR      HITCHCOCK A.       NORTH BY NORTHWEST
          R       CRONENBERG D.      DEAD RINGERS
          R       LUMET S.           MORNING AFTER, THE
          R       HITCHCOCK A.       PSYCHO
          PG13    HITCHCOCK A.       BIRDS, THE

                                                                    MORE
```

Press Enter to display the second screen of the report:

```
PAGE     2


CATEGORY  RATING  DIRECTOR            TITLE
--------  ------  --------            -----
MYSTERY   R       BECKER H.           SEA OF LOVE
```

```
                              END OF REPORT
```

Issue the FIN command to exit FOCUS:

```
FIN
```

Once you issue the FIN command, you are no longer in FOCUS, so recording stops. However, if you want to invoke FOCUS at this point without recording the new session, you must free the FOCREPLAY FILEDEFs. If you do not, the new session's script will be appended to the existing script file and its output will be appended to the existing output file.

Example: Recording a Session Under z/OS

You can create the two files needed for FOCREPLAY input mode prior to executing the CLIST used to invoke FOCUS or in the CLIST. If you create these files in the CLIST, you must then free the allocations and reallocate them with DISP MOD. For example, the following commands allocate new data sets to contain the session script and the session output:

```
ALLOC F(FSC3270I) NEW DA('USER1.FSC.SCRIPT') ,
                 SPACE(50,10) TRACKS LRECL(23472) BLKSIZE(23476) ,
                 DSORG(PS) RECFM(V)
ALLOC F(FSC3270O) NEW DA('USER1.FSC.OUTBASE') ,
                 SPACE(50,10) TRACKS LRECL(23472) BLKSIZE(23476) ,
                 DSORG(PS) RECFM(V)"
```

The following commands free these allocations and reallocate the data sets with DISP MOD:

```
FREE F(FSC3270I)
FREE F(FSC3270O)
ALLOC F(FSC3270I) REUSE MOD DA('USER1.FSC.SCRIPT')
ALLOC F(FSC3270O) REUSE MOD DA('USER1.FSC.OUTBASE')
```

The following FOCUS session issues a TABLE request against the MOVIES data source. This session is recorded by FOCREPLAY as a result of executing the CLIST containing the allocations for DDNAMEs FSC3270I and FSC3270O.

Issue the following TABLE request:

```
>  > TABLE FILE MOVIES
> PRINT RATING DIRECTOR TITLE/A20
> BY CATEGORY
> WHERE CATEGORY EQ 'DRAMA' OR 'MYSTERY' OR 'CHILDREN'
> END

 NUMBER OF RECORDS IN TABLE=       18  LINES=     18

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
```

Press Enter to display the first screen of the report:

```
PAGE     1


CATEGORY  RATING  DIRECTOR           TITLE
--------  ------  --------           -----
CHILDREN  NR                         SMURFS, THE
          G       BARTON C.          SHAGGY DOG, THE
          NR                         SCOOBY-DOO-A DOG IN
          G       GEROMINI           ALICE IN WONDERLAND
          NR                         SESAME STREET-BEDTIM
          NR                         ROMPER ROOM-ASK MISS
          NR      DISNEY W.          SLEEPING BEAUTY
          G       DISNEY W.          BAMBI
DRAMA     R       LUMET S.           DOG DAY AFTERNOON
MYSTERY   PG      HITCHCOCK A.       REAR WINDOW
          PG      HITCHCOCK A.       VERTIGO
          R       GRANT M.           FATAL ATTRACTION
          NR      HITCHCOCK A.       NORTH BY NORTHWEST
          R       CRONENBERG D.      DEAD RINGERS
          R       LUMET S.           MORNING AFTER, THE
          R       HITCHCOCK A.       PSYCHO
          PG13    HITCHCOCK A.       BIRDS, THE

                                                               MORE
```

Press Enter to display the second screen of the report:

```
PAGE      2


CATEGORY  RATING  DIRECTOR            TITLE
--------  ------  --------            -----
MYSTERY   R       BECKER H.           SEA OF LOVE



























                              END OF REPORT
```

Issue the FIN command to exit FOCUS:

```
FIN
```

Once you issue the FIN command, you are no longer in FOCUS, so recording stops. However, if you want to invoke FOCUS at this point without recording the new session, you must free the FOCREPLAY allocations. If you do not, the new session's script will be appended to the existing script file and its output will be appended to the existing output file.

Replaying a Recorded FOCUS Session

How to:

Replay a Recorded FOCUS Session Interactively

Reference:

FOCREPLAY PFKeys

Example:

Replaying a Recorded FOCUS Session Interactively Under z/VM

Replaying a Recorded FOCUS Session Interactively Under z/OS

You can replay a recorded FOCUS session interactively or in a batch job. **Note:** Do *not* allocate DDNAMEs SYSIN or SYSPRINT in the FOCREPLAY batch job.

If you replay the session interactively, you must press a key in order to proceed from one line to the next in the script. Press Enter to move through the script line by line or, if your FOCUS session is accepting input, you can press a PFkey to move more quickly through the script.

Procedure: How to Replay a Recorded FOCUS Session Interactively

1. Allocate or FILEDEF the required DDNAMES in your CLIST that invokes FOCUS. Allocate:
   - ❏ DDNAME FSC3270Q to the file containing the previously recorded script.
   - ❏ DDNAME FSC3270O to the file to receive the replayed session output.
   - ❏ DDNAME FSC3270I as DUMMY
2. Execute your REXX EXEC or CLIST that invokes FOCUS.

   The first line in the script displays on the screen.
3. Press Enter to invoke the next line in the script or, if your FOCUS session is accepting input, you can press a PFkey to move more quickly through the script.
4. Continue pressing Enter or a PFkey to move through the script.

   When the FIN command is replayed, the FOCUS session ends, and the replay stops.

   Note: If you want to invoke FOCUS at this point without replaying the session again, you must clear the FOCREPLAY FILEDEFs or free the FOCREPLAY allocations. If you do not, the new session output will be appended to the existing output file.

Reference: FOCREPLAY PFKeys

If your FOCUS session is accepting input, you can press the Enter key to step through the session line by line, or you can press one of the following PFkeys:

PFkey	Action
F1	Plays the next 10 script lines with no confirmation.
F2	Plays the next 20 script lines with no confirmation.
F3	Plays the next 30 script lines with no confirmation.
F4	Plays the next 40 script lines with no confirmation.
F5	Plays the next 50 script lines with no confirmation.
F6	Plays the next 60 script lines with no confirmation.
F7	Plays the next 70 script lines with no confirmation.
F8	Plays the next 80 script lines with no confirmation.
F9	Plays the next 90 script lines with no confirmation.
F10	Plays the next 100 script lines with no confirmation.
F11	Plays to the end of the script or to a break point with no confirmations.
F12	Quit (exits FOCUS by issuing Abend U610-1). At this point you must log off and on again to clean up after the abend.
F24	Exits to a live FOCUS session. If you are re-recording a session (by allocating an input file), you can input new key strokes at this point.

Example: Replaying a Recorded FOCUS Session Interactively Under z/VM

The REXX EXEC that invokes FOCUS has the following FOCREPLAY FILEDEFs:

- ❑ The previously recorded script:

```
'FILEDEF FSC3270Q DISK FSSCRIPT IN A'
```

- ❑ The output of the replayed session:

```
'FILEDEF FSC3270O DISK FSC3270 OUTNEW A (DISP MOD'
```

- ❑ The dummy FILEDEF for DDNAME FSC3270I:

```
'FILEDEF FSC3270I DUMMY'
```

Once the REXX EXEC executes and invokes FOCUS, the first line of the script file displays:

```
>  >  TABLE FILE MOVIES
```

You can press Enter, or you can press one of the FOCREPLAY PFkeys described in *FOCREPLAY PFKeys*.

For this example, press Enter to execute this line. The next line displays:

```
>  PRINT RATING DIRECTOR TITLE/A20
```

Press Enter to execute this line. The next line displays:

```
>  BY CATEGORY
```

Press Enter to execute this line. The next line displays:

```
>  WHERE CATEGORY EQ 'DRAMA' OR 'MYSTERY' OR 'CHILDREN'
```

Press Enter to execute this line. The next line displays:

```
 >  END
```

Press Enter to execute this line. The report summary lines display:

```
NUMBER OF RECORDS IN TABLE=       18  LINES=     18

PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
```

Press Enter. The first screen of report output displays:

```
PAGE     1


CATEGORY  RATING  DIRECTOR           TITLE
--------  ------  --------           -----
CHILDREN  NR                         SMURFS, THE
          G       BARTON C.          SHAGGY DOG, THE
          NR                         SCOOBY-DOO-A DOG IN
          G       GEROMINI           ALICE IN WONDERLAND
          NR                         SESAME STREET-BEDTIM
          NR                         ROMPER ROOM-ASK MISS
          NR      DISNEY W.          SLEEPING BEAUTY
          G       DISNEY W.          BAMBI
DRAMA     R       LUMET S.           DOG DAY AFTERNOON
MYSTERY   PG      HITCHCOCK A.       REAR WINDOW
          PG      HITCHCOCK A.       VERTIGO
          R       GRANT M.           FATAL ATTRACTION
          NR      HITCHCOCK A.       NORTH BY NORTHWEST
          R       CRONENBERG D.      DEAD RINGERS
          R       LUMET S.           MORNING AFTER, THE
          R       HITCHCOCK A.       PSYCHO
          PG13    HITCHCOCK A.       BIRDS, THE

                                                                        MORE
```

Press Enter. The next screen of report output displays:

```
PAGE     2


CATEGORY  RATING  DIRECTOR           TITLE
--------  ------  --------           -----
MYSTERY   R       BECKER H.          SEA OF LOVE










                                END OF REPORT
```

Press Enter. The next line of the script displays:

```
>  >  FIN
```

Press Enter to execute this line.

Once the FIN command executes, you exit FOCUS, so replay ends. However, if you want to invoke FOCUS at this point without replaying the session again, you must clear the FOCREPLAY FILEDEFs. If you do not, the new session output will be appended to the existing output file.

After replaying a session, you have one output file from the recorded session and another output file from the replayed session. You can compare these two output files.

Example: Replaying a Recorded FOCUS Session Interactively Under z/OS

The CLIST that invokes FOCUS has the following FOCREPLAY allocations:

- The previously recorded script:

  ```
  ALLOC F(FSC3270Q) DA ('USER1.FSC.SCRIPT') MOD
  ```

- The output of the replayed session:

  ```
  ALLOC F(FSC3270O) DA ('USER1.FSC.OUTNEW') MOD
  ```

- The dummy allocation for DDNAME FSC3270I:

  ```
  ALLOC F(FSC3270I) DUMMY
  ```

Once the CLIST executes, the first line of the script file displays:

```
>  >  TABLE FILE MOVIES
```

You can press Enter, or you can press one of the FOCREPLAY PFkeys described in *FOCREPLAY PFKeys*.

For this example, press Enter to execute this line. The next line displays:

```
>  PRINT RATING DIRECTOR TITLE/A20
```

Press Enter to execute this line. The next line displays:

```
>  BY CATEGORY
```

Press Enter to execute this line. The next line displays:

```
>  WHERE CATEGORY EQ 'DRAMA' OR 'MYSTERY' OR 'CHILDREN'
```

Press Enter to execute this line. The next line displays:

```
 >  END
```

Press Enter to execute this line. The report summary lines display:

```
NUMBER OF RECORDS IN TABLE=       18  LINES=     18

PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
```

Press Enter. The first screen of report output displays:

```
PAGE     1


CATEGORY  RATING  DIRECTOR           TITLE
--------  ------  --------           -----
CHILDREN  NR                         SMURFS, THE
          G       BARTON C.          SHAGGY DOG, THE
          NR                         SCOOBY-DOO-A DOG IN
          G       GEROMINI           ALICE IN WONDERLAND
          NR                         SESAME STREET-BEDTIM
          NR                         ROMPER ROOM-ASK MISS
          NR      DISNEY W.          SLEEPING BEAUTY
          G       DISNEY W.          BAMBI
DRAMA     R       LUMET S.           DOG DAY AFTERNOON
MYSTERY   PG      HITCHCOCK A.       REAR WINDOW
          PG      HITCHCOCK A.       VERTIGO
          R       GRANT M.           FATAL ATTRACTION
          NR      HITCHCOCK A.       NORTH BY NORTHWEST
          R       CRONENBERG D.      DEAD RINGERS
          R       LUMET S.           MORNING AFTER, THE
          R       HITCHCOCK A.       PSYCHO
          PG13    HITCHCOCK A.       BIRDS, THE

                                                              MORE
```

Press Enter. The next screen of report output displays:

```
PAGE     2


CATEGORY  RATING  DIRECTOR            TITLE
--------  ------  --------            -----
MYSTERY   R       BECKER H.           SEA OF LOVE
















                               END OF REPORT
```

Press Enter. The next line of the script displays:

```
>  >  FIN
```

Press Enter to execute this line.

Once the FIN command executes, you exit FOCUS, so replay ends. However, if you want to invoke FOCUS at this point without replaying the session again, you must free the FOCREPLAY allocations. If you do not, the new session output will be appended to the existing output file.

After replaying a session, you have one output file from the recorded session and another output file from the replayed session. You can compare these two output files.

Comparing a Recorded Session With a Replayed Session

Example:

Comparing Two Sessions Using the XCOMPARE Facility on z/VM

Comparing Two Sessions Using the ISPF SuperCE Utility

FOCREPLAY does not compare output files. You can use any file comparison tool installed at your site to perform the comparison. When you view the results, you must determine which differences are acceptable and do not indicate a change in your application's operation.

Example: Comparing Two Sessions Using the XCOMPARE Facility on z/VM

Issue the following command at the Ready prompt:

```
XCOMPARE FSC3270 OUTBASE A FSC3270 OUTNEW A (DISK
```

The result of the comparison is placed in the file XCOMPARE LISTING A1:

```
1COMPARING 'FSC3270  OUTBASE  A1' WITH 'FSC3270  OUTNEW   A1'
0FILE 1       1  O: FOCUS  7.3.7   06/06/2006  16.06.11  GEN01.01
 FILE 2       1  O: FOCUS  7.3.7   06/06/2006  16.18.02  GEN01.01
0FILE 1      36  I00007: {Enter}{024-002}
0FILE 2      60  I00007: {Enter}{024-002}
0FILE 1      68  I00008: {Enter}{024-002}
0FILE 2      92  I00008: {Enter}{024-002}
0COMPARISON COMPLETED - RETURN CODE IS     8.
```

The differences in the two files are the lines with the date and time stamp and lines inserted by FOCREPLAY. These differences are acceptable and do not indicate a change in the operation of the application.

Example: Comparing Two Sessions Using the ISPF SuperCE Utility

Go to option 3.13 (the SuperCE Utility) from the ISPF Primary Option Menu.

Enter the names of the two output files on the screen and press Enter to start the comparison. This example accepts the default comparison options:

```
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
                               SuperCE Utility
Command ===>

New DS Name   . . . 'USER1.FSC.OUTBASE'
Old DS Name   . . . 'USER1.FSC.OUTNEW'
PDS Member List              (blank/pattern - member list, * - compare all)
   (Leave New/Old DSN "blank" for concatenated-uncataloged-password panel)
   Compare Type               Listing Type               Display Output
   2  1. File                 2  1. OVSUM                1  1. Yes
      2. Line                    2. Delta                   2. No
      3. Word                    3. CHNG                    3. Cond
      4. Byte                    4. Long                    4. UPD

Listing DSN  . . . . SUPERC.LIST
```

The results are placed in a file named USER1.SUPERC.LIST and are automatically displayed when the comparison is complete. In this example, the only lines that are different are those that show the date and time of the two runs. This difference does not indicate a change in the operation of the application:

```
                     LISTING OUTPUT SECTION (LINE COMPARE)

ID       SOURCE LINES

I - O:   FOCUS  7.3.7   06/07/2006  15.46.14
D - O:   FOCUS  7.3.7   06/07/2006  15.48.43


                   LINE COMPARE SUMMARY AND STATISTICS

 106 NUMBER OF LINE MATCHES              1  TOTAL CHANGES (PAIRED+NONPAIRED CHNG)
   0 REFORMATTED LINES                   1  PAIRED CHANGES (REFM+PAIRED INS/DEL)
   1 NEW FILE LINE INSERTIONS            0  NON-PAIRED INSERTS
   1 OLD FILE LINE DELETIONS             0  NON-PAIRED DELETES
 107 NEW FILE LINES PROCESSED
 107 OLD FILE LINES PROCESSED

LISTING-TYPE = DELTA      COMPARE-COLUMNS =    1:23468     LONGEST-LINE = 126
PROCESS OPTIONS USED: NONE

 ISRS004I LISTING LINES MAY BE TRUNCATED DUE TO LIMITING OUTPUT LINE WIDTH.
```

Stopping Replay at a Break Point

If you want to stop the replay at a break point, edit the input file and add the following on a line by itself right in front of the last line you want executed before the break point:

```
<BREAK>
```

On replay (online only), the input sequence is replayed up to the line following the break point, and then the keyboard is opened for stepwise motion through the remainder of the script using existing PF key functions. This technique enables you get to a specific point in a long script very quickly.

If you use F10 or F11 to move down in the script, and a break is encountered along the way, replay stops at the break point rather than proceeding to where the PF key would have taken you.

You can set multiple break points in a script file by inserting multiple <BREAK> commands in the file.

Re-recording From the Middle or End of a Script

In order to continue recording a partially recorded session, you must allocate DDNAME FSC3270I to a file that will contain the new version of the script. The existing script must be in a different file, allocated to DDNAME FSC3270Q. In addition, if you allocate an existing output file to DDNAME FSC3270O (to receive the session output), the new output will appended to the existing output. If you want the re-recorded session to go to a separate file, allocate a new output file to DDNAME FSC3270O.

To start recording where you left off last time, remove the FIN command and execute replay mode online. The existing script will be re-recorded to the new script file. At the end of the existing script, the keyboard will open for input, and your key strokes will be recorded at the end of the new script file.

To start recording in the middle of a script file, edit the existing script file and add the following on a line by itself at the point in the script at which you want to start re-recording:

```
<RECORD>
```

In an online session, the replay tool plays the existing script and re-records it to the new script file. When it encounters the <RECORD> command, it opens the keyboard for further input that will be recorded and added to the end of the new script file.

This technique eliminates trailing lines after a mistake and enables you to re-record the remainder of the session.

In this way, you do not have to start recording from scratch when a mistake is made during the recording session or when you stop at some point but want to continue recording later.

You must allocate the new script file to DDNAME FSC3270Q when replaying the corrected or extended session.

&FOCFEXNAME Variable

The Dialogue Manager variable &FOCFEXNAME returns the name of the FOCEXEC running even if it was executed using an EX command or a -INCLUDE command from within another FOCEXEC. This variable differs from the &FOCFOCEXEC variable because &FOCFOCEXEC returns the name of the calling FOCEXEC only.

Example: Retrieving the Name of the Active FOCEXEC

The following request consists of three procedures.

The main procedure, FEXNAME1, types the values of the variables &FOCFOCEXEC and &FOCFEXNAME and then includes the procedure called FEXNAME3:

```
-TYPE THIS IS FOCEXEC NUMBER 1
 -TYPE FOCFOCEXEC = &FOCFOCEXEC
 -TYPE FOCFEXNAME = &FOCFEXNAME
 -TYPE
 -INCLUDE FEXNAME3
```

The procedure FEXNAME3, includes the procedure called FEXNAME2 and then types the values of the variables &FOCFOCEXEC and &FOCFEXNAME:

```
TABLE FILE EMPLOYEE
 -INCLUDE FEXNAME2
 -TYPE THIS IS FOCEXEC 3
 -TYPE FOCFOCEXEC = &FOCFOCEXEC
 -TYPE FOCFEXNAME = &FOCFEXNAME
 -TYPE
 PRINT CURR_JOBCODE
 BY DEPARTMENT
 BY LAST_NAME
 BY FIRST_NAME
 END
```

The procedure FEXNAME2, types the values of the variables &FOCFOCEXEC and &FOCFEXNAME:

```
-TYPE THIS IS FOCEXEC 2
 -TYPE FOCFOCEXEC = &FOCFOCEXEC
 -TYPE FOCFEXNAME = &FOCFEXNAME
 -TYPE
 HEADING CENTER
 "EMPLOYEES BY DEPARTMENT"
 ""
```

The output shows that the variable &FOCFOCEXEC always returns the name of the main procedure, while &FOCFEXNAME returns the name of the active procedure whether it is the main procedure or an included procedure:

```
EX FEXNAME1
```

```
THIS IS FOCEXEC NUMBER 1
 FOCFOCEXEC = FEXNAME1
 FOCFEXNAME = FEXNAME1

 THIS IS FOCEXEC 2
 FOCFOCEXEC = FEXNAME1
 FOCFEXNAME = FEXNAME2

 THIS IS FOCEXEC 3
 FOCFOCEXEC = FEXNAME1
 FOCFEXNAME = FEXNAME3
```

Controlling Missing Values in Reformatted Fields

How to:

Control Missing Values in Reformatted Fields

Reference:

Usage Notes for SET COMPMISS

Example:

Controlling Missing Values in Reformatted Fields

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

Syntax: How to Control Missing Values in Reformatted Fields

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

where:

ON

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

OFF

Displays a blank or zero for a reformatted field.

Note: The COMPMISS parameter cannot be set in an ON TABLE command.

Example: Controlling Missing Values in Reformatted Fields

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

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

The output is:

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

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

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

The output is:

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

Reference: Usage Notes for SET COMPMISS

If you create a HOLD file with COMPMISS ON, the HOLD Master File for the reformatted field indicates MISSING = ON (as does the original field). With COMPMISS = OFF, the reformatted field does NOT have MISSING = ON in the generated Master File.

Controlling Case Sensitivity of Passwords

How to:

Control Password Case Sensitivity

Example:

Controlling Password Case Sensitivity

When a DBA or user issues the SET USER, SET PERMPASS or SET PASS command, this user ID is validated before they are given access to any data source whose Master File has DBA attributes. The password is also checked when encrypting or decrypting a FOCEXEC.

The SET DBACSENSITIV command determines whether the password is converted to upper case prior to validation.

Syntax: How to Control Password Case Sensitivity

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

where:

`ON`

Does not convert passwords to upper case. All comparisons between the password set by the user and the password in the Master File or FOCEXEC are case sensitive.

`OFF`

Converts passwords to upper case prior to validation. All comparisons between the password set by the user and the password in the Master File or FOCEXEC are *not* case sensitive. OFF is the default value.

Example: Controlling Password Case Sensitivity

Consider the following DBA declaration added to the EMPLOYEE Master File:

```
USER = User2, ACCESS = RW,$
```

User2 wants to report from the EMPLOYEE data source and issues the following command:

```
SET USER = USER2
```

With DBACSENSITIV OFF, User2 can run the request even though the case of the password entered does not match the case of the password in the Master File.

With DBACSENSITIV ON, User2 gets the following message:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE:
```

With DBACSENSITIV ON, the user must issue the following command:

```
SET USER = User2
```

Note: In FOCUS for Mainframe, all user input is transmitted in upper case. Therefore, a mixed case password cannot be issued at the command line. It must be set in a FOCEXEC or profile.

SLEEP Function

How to:

Suspend Execution for a Specified Number of Seconds

Example:

Suspending Execution for Four Seconds

As of FOCUS 7.6.3, you can suspend FOCUS execution for a specified number of seconds.

The SLEEP function suspends execution for the number of seconds you specify as its input argument.

This function is most useful in Dialogue Manager when you need to wait to start a specific procedure. For example, you can start a FOCUS Database Server and wait until the server is started before initiating a client application.

Syntax: How to Suspend Execution for a Specified Number of Seconds

```
SLEEP(delay, outfld);
```

where:

delay

Numeric

Is the number of seconds to delay execution. The number can be specified down to the millisecond.

outfld

Numeric

Is the name of a field or a format enclosed in single quotation marks. The value returned is the same value you specify for *delay*.

Example: Suspending Execution for Four Seconds

The following example computes the current date and time, suspends execution for 4 seconds, and computes the current date and time after the delay:

```
TABLE  FILE VIDEOTRK
PRINT TRANSDATE NOPRINT
COMPUTE
START_TIME/HYYMDSa = HGETC(8, START_TIME);
DELAY/I2 = SLEEP(4.0, 'I2');
END_TIME/HYYMDSa = HGETC(8, END_TIME);
IF RECORDLIMIT EQ 1
END
```

The output is:

```
START_TIME              DELAY  END_TIME
----------              -----  --------
2007/10/26  5:04:36pm       4  2007/10/26  5:04:40pm
```

Setting a Currency Symbol for the M and N Format Options

As of FOCUS 7.6.4, SET CURRSYMB determines which currency symbol will be displayed when a format specification uses the M and N edit options.

Syntax: How to Specify a Currency Symbol

```
SET CURRSYMB = symbol
```

In a request, use

```
ON TABLE SET CURRSYMB symbol
```

where:

symbol

Is any printable character or a supported currency code. **Note:** In order to specify a dollar sign as the character, you must enclose it in single quotation marks ('$'). The following are supported currency codes:

USD or '$' specifies U. S. dollars.

GBP specifies the British pound.

JPY specifies the Japanese yen.

EUR specifies the Euro.

Example: Displaying the Japanese Yen Symbol

The following request uses the EMPLOYEE data source and sets the currency symbol to the Japanese yen. Since the EMPLOYEE Master File specifies the format D12.2M for the CURR_SAL field, the yen symbol displays as the currency symbol for its field values:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY LAST_NAME
ON TABLE SET CURRSYMB JPY
ON TABLE COLUMN-TOTAL
END
```

The output is:

```
LAST_NAME                CURR_SAL
---------                --------
BANNING                ¥29,700.00
BLACKWOOD              ¥21,780.00
CROSS                  ¥27,062.00
GREENSPAN               ¥9,000.00
IRVING                 ¥26,862.00
JONES                  ¥18,480.00
MCCOY                  ¥18,480.00
MCKNIGHT               ¥16,100.00
ROMANS                 ¥21,120.00
SMITH                  ¥13,200.00
                        ¥9,500.00
STEVENS                ¥11,000.00

TOTAL                 ¥222,284.00
```

ISO Standard Week Numbering and the HYYWD Function

In this section:

HYYWD Function: Returning the Year and Week Number From a Date-time Value

How to:

Specify the First Day of the Week

Example:

Extracting the Week Component From a Date-Time Value

Starting in Version 7.6.5, you can extract week numbers from date-time values using the ISO 8601 standard definition.

The HPART and HNAME subroutines can extract a week number from a date-time value. To determine a week number, they can use ISO 8601 standard week numbering, which defines the first week of the year as the first week in January with four or more days. Any preceding days in January belong to week 52 or 53 of the preceding year. The ISO standard also establishes Monday as the first day of the week.

These functions can also define the first week of the year as the first week in January with seven days. This is the definition they used in prior releases.

You specify which type of week numbering to use by setting the WEEKFIRST parameter.

Since the week number returned by HNAME and HPART functions can be in the current year or the year preceding or following, the week number by itself may not be useful. The function HYYWD returns both the year and the week from a given date-time value.

Syntax: How to Specify the First Day of the Week

```
SET WEEKFIRST = value
```

where:

value

Can be:

1 through 7, representing Sunday through Saturday with non-standard week numbering

or

ISO1 through ISO7, representing Sunday through Saturday with ISO standard week numbering. **Note:** ISO is a synonym for ISO2.

The ISO standard establishes Monday as the first day of the week, so to be fully ISO compliant, the WEEKFIRST parameter should be set to ISO or ISO2.

Example: Extracting the Week Component From a Date-Time Value

In the following request, HNAME extracts the week in alphanumeric format from the TRANSDATE field. Changing the WEEKFIRST setting changes the value of the extracted component.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
WEEK_COMPONENT/A10 = HNAME(TRANSDATE, 'WEEK', 'A10');
WHERE DATE EQ 1991 AND QUANTITY GT 1
END
```

When WEEKFIRST is set to 2, the output is:

```
CUSTID  DATE-TIME         WEEK_COMPONENT
------  ---------         --------------
0925    1991/06/27 02:45  25
1297    1991/06/24 04:43  25
1423    1991/06/25 01:17  25
```

When WEEKFIRST is set to ISO2, the output is:

```
CUSTID  DATE-TIME         WEEK_COMPONENT
------  ---------         --------------
0925    1991/06/27 02:45  26
1297    1991/06/24 04:43  26
1423    1991/06/25 01:17  26
```

HYYWD Function: Returning the Year and Week Number From a Date-time Value

How to:

Return the Year and Week Number From a Date-time Value

Example:

Returning the Year and Week Number From a Date-time Value

Extracting a Component From a Date Returned by HYYWD

The week number returned by HNAME and HPART can actually be in the year preceding or following the input date.

The HYYWD function returns both the year and the week number from a given date-time value.

The output is edited to conform to the ISO standard format for dates with week numbers, yyyy-Www-d.

Syntax: How to Return the Year and Week Number From a Date-time Value

```
HYYWD(dtvalue, outfield)
```

where:

dtvalue

Date-time

Is the date-time value to be edited, the name of a date-time field that contains the value, or an expression that returns the value.

outfield

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

The field must be at least 10 characters long. The output is in the following format:

`yyyy-Www-d`

where:

`yyyy`

Is the four-digit year.

`ww`

Is the two-digit week number (01 to 53).

`d`

Is the single-digit day of the week (1 to 7). The *d* value is relative to the current WEEKFIRST setting. If WEEKFIRST is 2 or ISO2 (Monday), then Monday is represented in the output as 1, Tuesday as 2.

Using the EDIT function, you can extract the individual subfields from this output.

Example: Returning the Year and Week Number From a Date-time Value

The following request against the VIDEOTR2 data source calls HYYWD to convert the TRANSDATE date-time field to the ISO standard format for dates with week numbers. WEEKFIRST is set to ISO2, which produces ISO standard week numbering:

```
SET WEEKFIRST = ISO2
TABLE FILE VIDEOTR2
SUM TRANSTOT QUANTITY
COMPUTE ISODATE/A10 = HYYWD(TRANSDATE, 'A10');
BY TRANSDATE
WHERE QUANTITY GT 1
END
```

The output is:

```
TRANSDATE          TRANSTOT  QUANTITY  ISODATE
---------          --------  --------  -------
1991/06/24 04:43      16.00         2  1991-W26-1
1991/06/25 01:17       2.50         2  1991-W26-2
1991/06/27 02:45      16.00         2  1991-W26-4
1996/08/17 05:11       5.18         2  1996-W33-6
1998/02/04 04:11      12.00         2  1998-W06-3
1999/01/30 04:16      13.00         2  1999-W04-6
1999/04/22 06:19       3.75         3  1999-W16-4
1999/05/06 05:14       1.00         2  1999-W18-4
1999/08/09 03:17      15.00         2  1999-W32-1
1999/09/09 09:18      14.00         2  1999-W36-4
1999/10/16 09:11       5.18         2  1999-W41-6
1999/11/05 11:12       2.50         2  1999-W44-5
1999/12/09 09:47       5.18         2  1999-W49-4
1999/12/15 04:04       2.50         2  1999-W50-3
```

Example: Extracting a Component From a Date Returned by HYYWD

The following request against the VIDEOTR2 data source calls HYYWD to convert the TRANSDATE date-time field to the ISO standard format for dates with week numbers. It then uses the EDIT function to extract the week component from this date. WEEKFIRST is set to ISO2, which produces ISO standard week numbering:

```
SET WEEKFIRST = ISO2
TABLE FILE VIDEOTR2
SUM TRANSTOT QUANTITY
COMPUTE ISODATE/A10 = HYYWD(TRANSDATE, 'A10');
COMPUTE WEEK/A2 = EDIT(ISODATE, '$$$$$$99$$');
BY TRANSDATE
WHERE QUANTITY GT 1 AND DATE EQ 1991
END
```

The output is:

```
TRANSDATE          TRANSTOT  QUANTITY  ISODATE     WEEK
---------          --------  --------  -------     ----
1991/06/24 04:43      16.00         2  1991-W26-1  26
1991/06/25 01:17       2.50         2  1991-W26-2  26
1991/06/27 02:45      16.00         2  1991-W26-4  26
```

PATTERN Function

How to:

Generate a Pattern From an Input String

Example:

Producing a Pattern From Alphanumeric Data

Starting in Version 7.6.5, you can examine a string to see if it follows a standard pattern.

The PATTERN function examines a source string and produces a pattern that indicates the sequence of numbers, uppercase letters, and lowercase letters in the input string. In the output pattern, any character from the input that represented a digit becomes the character ‘9’, any character that represents an uppercase letter becomes ‘A’, and any character that represents a lowercase letter becomes ‘a’. Special characters remain unchanged. An unprintable character becomes the character ‘X’. This function is useful for examining data to make sure that it follows a standard pattern.

Syntax: How to Generate a Pattern From an Input String

```
PATTERN (length, infield, outfield)
```

where:

infield

Alphanumeric

Is a field containing the input string, or a literal string enclosed in single quotation marks.

length

Numeric

Is the length of *infield*.

outfield

Alphanumeric

Is the name of the field to contain the result or the format of the field enclosed in single quotation marks.

Example: Producing a Pattern From Alphanumeric Data

The following 19 records are stored in a fixed format sequential file (with LRECL 14) named TESTFILE:

```
212-736-6250
212 736 4433
123-45-6789
800-969-INFO
10121-2898
10121
2 Penn Plaza
917-339-6380
917-339-4350
(212) 736-6250
(212) 736-4433
212-736-6250
212-736-6250
212-736-6250
(212) 736 5533
(212) 736 5533
(212) 736 5533
10121 Æ
800-969-INFO
```

The Master File is:

```
FILENAME=TESTFILE, SUFFIX=FIX     ,
  SEGMENT=TESTFILE, SEGTYPE=S0, $
    FIELDNAME=TESTFLD, USAGE=A14, ACTUAL=A14, $
```

The following request generates a pattern for each instance of TESTFLD and displays them by the pattern that was generated. It shows the count of each pattern and its percentage of the total count. The PRINT command shows which values of TESTFLD generated each pattern.

```
CMS FILEDEF TESTFILE DISK TESTFILE FOCTEMP A
DEFINE FILE TESTFILE
   PATTERN/A14 = PATTERN (14, TESTFLD, 'A14' ) ;
END

TABLE FILE TESTFILE
  SUM CNT.PATTERN AS 'COUNT' PCT.CNT.PATTERN AS 'PERCENT'
   BY PATTERN
 PRINT TESTFLD
   BY PATTERN
ON TABLE COLUMN-TOTAL
END
```

Note that the next to last line produced a pattern from an input string that contained an unprintable character, so that character was changed to X. Otherwise, each numeric digit generated a 9 in the output string, each uppercase letter generated the character ‘A’, and each lowercase letter generated the character ‘a’. The output is:

```
PATTERN          COUNT  PERCENT  TESTFLD
-------          -----  -------  -------
(999) 999 9999       3    15.79  (212) 736 5533
                                 (212) 736 5533
                                 (212) 736 5533
(999) 999-9999       2    10.53  (212) 736-6250
                                 (212) 736-4433
9 Aaaa Aaaaa         1     5.26  2 Penn Plaza
999 999 9999         1     5.26  212 736 4433
999-99-9999          1     5.26  123-45-6789
999-999-AAAA         2    10.53  800-969-INFO
                                 800-969-INFO
999-999-9999         6    31.58  212-736-6250
                                 917-339-6380
                                 917-339-4350
                                 212-736-6250
                                 212-736-6250
                                 212-736-6250
99999                1     5.26  10121
99999 X              1     5.26  10121 Æ
99999-9999           1     5.26  10121-2898

TOTAL               19   100.00
```

REVERSE Function

How to:

Reverse the Characters in a String

Example:

Reversing the Characters in a String

Starting in Version 7.6.5, you can use REVERSE to reverse the characters in a string.

The REVERSE function reverses the characters of a string. This reversal includes all trailing blanks, which then become leading blanks. However, in an HTML report with SET SHOWBLANKS=OFF (the default value), the leading blanks are not visible.

Syntax: How to Reverse the Characters in a String

`REVERSE(length, string, outfield)`

where:

length

Integer

Is the length in characters of *string* and *outfield*, or a field that contains the length.

string

Alphanumeric

Is the character string enclosed in single quotation marks, or a field that contains the character string.

outfield

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Example: Reversing the Characters in a String

In the following request against the EMPLOYEE data source, the REVERSE function is used to reverse the characters in the LAST_NAME field to produce the field named REVERSE_LAST. In this field, the trailing blanks from LAST_NAME have become leading blanks. The TRIM function is used to strip the leading blanks from REVERSE_LAST to produce the field named TRIM_REVERSE:

```
DEFINE FILE EMPLOYEE
REVERSE_LAST/A15 = REVERSE(15, LAST_NAME, REVERSE_LAST);
TRIM_REVERSE/A15 = TRIM('L', REVERSE_LAST, 15, ' ', 1, 'A15');
END

TABLE FILE EMPLOYEE
PRINT REVERSE_LAST TRIM_REVERSE
BY LAST_NAME
END
```

The output is:

```
LAST_NAME         REVERSE_LAST     TRIM_REVERSE
---------         ------------     ------------
BANNING                GNINNAB   GNINNAB
BLACKWOOD            DOOWKCALB   DOOWKCALB
CROSS                    SSORC   SSORC
GREENSPAN            NAPSNEERG   NAPSNEERG
IRVING                  GNIVRI   GNIVRI
JONES                    SENOJ   SENOJ
MCCOY                    YOCCM   YOCCM
MCKNIGHT              THGINKCM   THGINKCM
ROMANS                  SNAMOR   SNAMOR
SMITH                    HTIMS   HTIMS
                         HTIMS   HTIMS
STEVENS                SNEVETS   SNEVETS
```

XTPACK Function

How to:

Store Packed Values in an Alphanumeric Field

Example:

Writing a Long Packed Number to an Output File

Starting in Version 7.6.5, you can write packed fields from 1 to 16 bytes to an output file.

The XTPACK function stores packed numbers with up to 31 significant digits in an alphanumeric field, retaining decimal data. This permits writing a short or long packed field of any length, 1 to 16 bytes, to an output file.

Syntax: How to Store Packed Values in an Alphanumeric Field

```
XTPACK (infield, outlength, outdec, outfield)
```

where:

infield

Numeric

Is the field that contains the packed value.

outlength

Numeric

Is the length of the alphanumeric field that will hold the converted packed field. Can be from 1 to 16.

`outdec`

Numeric

Is the number of decimal positions for *outfield*.

`outfield`

Alphanumeric

Is the name of the field to contain the result or the format of the field enclosed in single quotation marks.

Example: Writing a Long Packed Number to an Output File

The following request creates a long packed decimal field named LONGPCK that has format P25.2 and is created by adding the number 1111111111111111111 to PCT_INC. ALPHAPCK (format A13) is the result of applying XTPACK to the long packed field. ALPHAPCK contains a packed number stored in an alphanumeric field and is not printable, so HEX_ALPHAPCK converts it to its printable hexadecimal equivalent using the UFMT subroutine. PCT_INC, LONGPCK, ALPHAPCK, and HEX_ALPHAPCK are then written to an alphanumeric HOLD file named XTOUT.

```
SET HOLDLIST = PRINTONLY
DEFINE FILE EMPLOYEE
LONGPCK/P25.2 = PCT_INC + 1111111111111111111;
ALPHAPCK/A13 = XTPACK(LONGPCK,13,2,'A13');
END
TABLE FILE EMPLOYEE
PRINT PCT_INC LONGPCK ALPHAPCK
  COMPUTE HEX_ALPHAPCK/A26 = UFMT(ALPHAPCK,13,'A26') ;
WHERE PCT_INC GT 0
  ON TABLE HOLD AS XTOUT FORMAT ALPHA
END
```

The following request prints the PCT_INC, LONGPCK, and HEX_ALPHAPCK fields from the XTOUT HOLD file (ALPHAPCK is not printable):

```
TABLE FILE XTOUT
PRINT PCT_INC LONGPCK HEX_ALPHAPCK
END
```

The output shows the that the HEX_ALPHAPCK field is the packed equivalent of the LONGPCK field. The sign is at the end of packed numbers. Note that a positive sign may be represented as either C or F, depending on operating environment:

```
PCT_INC                    LONGPCK  HEX_ALPHAPCK
-------                    -------  ------------
    .10    11111111111111111111.10  00011111111111111111111110C
    .12    11111111111111111111.12  00011111111111111111111112C
    .10    11111111111111111111.10  00011111111111111111111110C
    .04    11111111111111111111.04  00011111111111111111111104C
    .05    11111111111111111111.05  00011111111111111111111105C
    .10    11111111111111111111.10  00011111111111111111111110C
    .15    11111111111111111111.15  00011111111111111111111115C
    .07    11111111111111111111.07  00011111111111111111111107C
    .04    11111111111111111111.04  00011111111111111111111104C
    .05    11111111111111111111.05  00011111111111111111111105C
```

The Master File created as a result of the HOLD command follows. ALPHAPCK is described as A13:

```
FILENAME=XTOUT   , SUFFIX=FIX     , $
  SEGMENT=XTOUT, SEGTYPE=S0, $
    FIELDNAME=PCT_INC, ALIAS=E01, USAGE=F6.2, ACTUAL=A06, $
    FIELDNAME=LONGPCK, ALIAS=E02, USAGE=P25.2, ACTUAL=A25, $
    FIELDNAME=ALPHAPCK, ALIAS=E03, USAGE=A13, ACTUAL=A13, $
    FIELDNAME=HEX_ALPHAPCK, ALIAS=E04, USAGE=A26, ACTUAL=A26, $
```

You can now create a new data source with the packed values written to the XTOUT HOLD file. In the HOLD file, ALPHAPCK is described as A13, but it contains packed values, not alphanumeric data. In the new data source to be created, those packed values will be loaded into the field named PACKEDVAL that is described as USAGE=P25.2, ACTUAL = P16. The field LONGPCK from the HOLD file contains the alphanumeric equivalent of the packed value, so it is loaded into ALPHAVAL in the new data source:

```
FILENAME=XTPACKIN, SUFFIX=FIX     , $
SEGMENT=XTPACKIN, SEGTYPE=S0, $
FIELDNAME=PCT_INC, ALIAS=E01, USAGE=F6.2, ACTUAL=F4 , $
FIELDNAME=ALPHAVAL, ALIAS=E02, USAGE=A25, ACTUAL=A25 , $
FIELDNAME=PACKEDVAL, ALIAS=E03, USAGE=P25.2, ACTUAL=P16, $
```

The following request creates the XTPACKIN data source and loads it with data from the HOLD file XTOUT. Note that the incoming packed transaction value from the HOLD file is described as P13:

```
CMS FILEDEF XTPACKIN DISK XTPACKIN FOCTEMP A
CREATE FILE XTPACKIN
-RUN
MODIFY FILE XTPACKIN
FIXFORM PCT_INC/6 ALPHAVAL/25 PACKEDVAL/P13
DATA ON XTOUT
END
```

The following request prints the values in the data source just created:

```
TABLE FILE XTPACKIN
PRINT *
END
```

The output shows that the long packed data was loaded and displays correctly. In the data source, the LONGPCK field contains the alphanumeric value and the ALPHAPCK field contains the packed equivalent:

```
PCT_INC  ALPHAVAL                                    PACKEDVAL
-------  ---------                                   ---------
    .10    11111111111111111111.10    11111111111111111111.10
    .12    11111111111111111111.12    11111111111111111111.12
    .10    11111111111111111111.10    11111111111111111111.10
    .04    11111111111111111111.04    11111111111111111111.04
    .05    11111111111111111111.05    11111111111111111111.05
    .10    11111111111111111111.10    11111111111111111111.10
    .15    11111111111111111111.15    11111111111111111111.15
    .07    11111111111111111111.07    11111111111111111111.07
    .04    11111111111111111111.04    11111111111111111111.04
    .05    11111111111111111111.05    11111111111111111111.05
```

Alternate Extended Currency Symbol

Reference:’

Extended Currency Symbol Formats

Starting in Version 7.6.5, the extended currency symbol format has been extended. The colon (:) has been added as an alternate for the exclamation point (!) for displaying extended currency symbols.

Extended currency symbol formats are specified as two-character combinations *in the last postition* of any numeric display format. The first character in the combination can be either an exclamation point (!) or a colon (:). The exclamation point is not consistent on all EBCDIC code pages and may produce unexpected behavior if the code page you are using translates the exclamation point differently:

Display Option	Description	Example
!d or :d	Fixed dollar sign.	D12.2:d
!D or :D	Floating dollar sign.	D12.2:D
!e or :e	Fixed euro symbol.	F9.2:e
!E or :E	Floating euro symbol on the left side.	F9.2:E
!F or :F	Floating euro symbol on the right side.	F9.2:F
!l or :l	Fixed British pound sign.	D12.1:l
!L or :L	Floating British pound sign.	D12.1:L
!y or :y	Fixed Japanese yen symbol.	I9:y
!Y or :Y	Floating Japanese yen symbol.	I9:Y

Reference: Extended Currency Symbol Formats

The following guidelines apply:

- ❏ A format specification cannot be longer than eight characters.
- ❏ The extended currency option must be the last option in the format.
- ❏ The extended currency symbol format cannot include the floating (M) or non-floating (N) display option.
- ❏ A non-floating currency symbol is displayed only on the first row of a report page. If you use field-based reformatting to display multiple currency symbols in a report column, only the symbol associated with the first row is displayed. In this case, do not use non-floating currency symbols.
- ❏ Lowercase letters are transmitted as uppercase letters by the terminal I/O procedures. Therefore, the fixed extended currency symbols can only be specified in a procedure.

- ❑ Extended currency symbol formats can be used with fields in floating point, decimal, packed, and integer formats. Alphanumeric and variable character formats cannot be used.

SET SUWEDGE

Starting in Version 7.6.5, keep master Files in memory on a sink machine.

The SET SUWEDGE command provides a mechanism for keeping Master files in memory on a FOCUS Database Server (sink machine) even though no users are accessing the files. This enhances performance by eliminating repeated parsing of Master files on FOCUS Database Servers that have a large number of users running multiple applications using different files on the Server.

You can specify that a number of files be wedged open, that specific named files be wedged open, or a combination of both. The maximum number of files that will be wedged open is 128, regardless of how many you specify.

Syntax: How to Wedge Master Files Open on a FOCUS Database Server

Place one or more of the following commands in the FOCUS Database Server profile (PROFILE HLI on z/VM, and member HLIPROF of the FOCEXEC data set on z/OS):

```
SET SUWEDGE = {n|ddname} [, {n|ddname} ...]
```

where:

n

Is a number of Master Files to be wedged open. You can specify any number, but once 128 files are wedged open, additional files will not be wedged. The default is zero.

ddname

Is the ddname of a specific file to be wedged open in addition to the *n* files specified by number (if any).

Note: To set both a number of files to be wedged and one or more specific DDNAMEs to be wedged, you can issue the SET SUWEDGE command mulitple times or specify multiple options in one SET SUWEDGE command. Specific DDNAMEs are not counted in the number of files specified (as long as the total does not exceed the maximum number of wedged files allowed). If you specify a number multiple times, the last one specified is the one used.

Reference: Usage Notes for SET SUWEDGE

- The most improvement in performance is gained by wedging open large Master Files and those used most often in MODIFY applications.
- Master Files are not opened by the SUWEDGE facility until a MODIFY or Maintain procedure that references them is executed.
- If a wedged Master File has a static cross reference to another Master File, only the parent Master File is wedged open when it is used in a MODIFY or Maintain application. A cross-referenced file is only wedged open when it is explicitly referenced in a MODIFY or Maintain procedure.
- All Master Files that participate in a COMBINE structure are opened when the COMBINE structure is referenced in a MODIFY application. Similarly, all files participating in a MAINTAIN procedure are opened when the Maintain procedure is executed.
- Using the SUWEDGE facility causes increased use of memory on the FOCUS Database Server.
- The maximum number of Master Files that can be open at one time on a FOCUS Database Server is 256, regardless of wedging.
- Files whose names begin with a number cannot be wedged open by explicitly specifying their names in the SET SUWEDGE command.
- If you specify a number of files to be wedged open and you issue a TABLE request on the FOCUS Database Server, the FOCUSSU file will be wedged open.
- You can see which files have been wedged open by examining the HLIPRINT file. The first OPN command listed for a wedged file will be associated with the user ID *suwedge*.

Example: Wedging Open Files on a FOCUS Database Server

The following commands wedge open any three FOCUS data sources plus the EMPLOYEE and CAR data sources:

```
SET SUWEDGE = 3
SET SUWEDGE = EMPLOYEE
SET SUWEDGE = CAR
```

or

```
SET SUWEDGE = 3
SET SUWEDGE = EMPLOYEE, CAR
```

or

```
SET SUWEDGE = EMPLOYEE, CAR
SET SUWEDGE = 3
```

or

```
SET SUWEDGE = 3, EMPLOYEE, CAR
```

5 Database Enhancements

This chapter described enhancements affecting databases and Master Files.

For additional information, see also:

- ❏ *Increased Number of Segments in a Structure* in Chapter 8, *Raised Limits*.
- ❏ *Increased Length for HOLD Files in FOCUS Format* in Chapter 8, *Raised Limits*.

Declaring Filters in a Master File

How to:

Declare a Filter in a Master File

Use a Master File Filter in a Request

Example:

Defining and Using a Master File Filter

Reference:

Usage Notes for Filters in a Master File

Boolean virtual fields (DEFINE fields that evaluate to TRUE or FALSE) can be used as record selection criteria. If the primary purpose of a virtual field is for use in record selection, you can clarify this purpose and organize virtual fields in the Master File by storing the expression using a FILTER declaration rather than a DEFINE. Filters offer the following features:

- ❑ They allow you to organize and store popular selection criteria in a Master File and reuse them in multiple requests and tools.
- ❑ For some data sources (such as VSAM and ISAM), certain filter expressions can be inserted inline into the WHERE or IF clause, enhancing optimization compared to a Boolean DEFINE.

Syntax: How to Declare a Filter in a Master File

```
FILTER  filtername = expression;
```

where:

filtername

Is the name assigned to the filter. The filter is internally assigned a format of I1, which cannot be changed.

expression

Is a logical expression that evaluates to TRUE (which assigns the value 1 to the filter field) or FALSE (which assigns the value 0 to the filter field). For any other type of expression, the field becomes a standard numeric virtual field in the Master File. Dialogue Manager variables (amper variables) can be used in the filter expression in same way they are used in standard Master File DEFINEs.

Syntax: How to Use a Master File Filter in a Request

```
TABLE FILE filename
   .
   .
   .
{WHERE|IF} expression_using_filters
```

where:

expression_using_filters

Is a logical expression that references a filter. In a WHERE phrase, the logical expression can reference one or more filters and/or virtual fields.

Reference: Usage Notes for Filters in a Master File

- ❏ The filter field name is internally assigned a format of I1 which cannot be changed.
- ❏ A filter can be used as a standard numeric virtual field anywhere in a report request, except that they are not supported in WHERE TOTAL tests.

Example: Defining and Using a Master File Filter

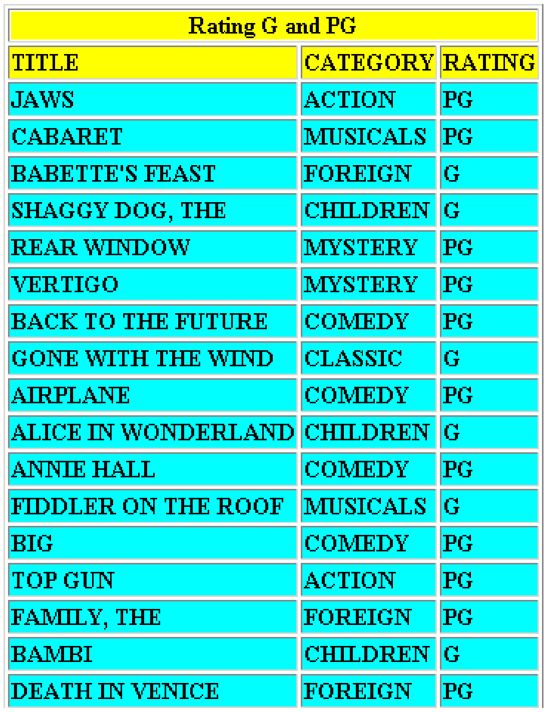
Consider the following filter declaration added to the MOVIES Master File:

```
FILTER G_RATING = RATING EQ 'G' OR 'PG'; $
```

The following request applies the G_RATING filter:

```
TABLE FILE MOVIES
HEADING CENTER
"Rating G and PG"
PRINT TITLE CATEGORY RATING
WHERE G_RATING
ON TABLE SET PAGE NOPAGE
ON TABLE SET GRID OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
type=report, style=bold, color=black, backcolor=yellow, $
type=data, backcolor=aqua, $
ENDSTYLE
END
```

The output is:



Rating G and PG		
TITLE	CATEGORY	RATING
JAWS	ACTION	PG
CABARET	MUSICALS	PG
BABETTE'S FEAST	FOREIGN	G
SHAGGY DOG, THE	CHILDREN	G
REAR WINDOW	MYSTERY	PG
VERTIGO	MYSTERY	PG
BACK TO THE FUTURE	COMEDY	PG
GONE WITH THE WIND	CLASSIC	G
AIRPLANE	COMEDY	PG
ALICE IN WONDERLAND	CHILDREN	G
ANNIE HALL	COMEDY	PG
FIDDLER ON THE ROOF	MUSICALS	G
BIG	COMEDY	PG
TOP GUN	ACTION	PG
FAMILY, THE	FOREIGN	PG
BAMBI	CHILDREN	G
DEATH IN VENICE	FOREIGN	PG

Describing GROUP Fields as a Set of Elements

How to:

Describe a GROUP Field as a Set of Elements

Reference:

Usage Notes for Group Elements

Example:

Declaring a GROUP With ELEMENTS

A GROUP declaration in a Master File describes several fields as a single entity. One use of a group is to describe group keys in a VSAM data source. Sometimes referring to several fields by one group name facilitates ease of reporting.

Traditionally, when describing a GROUP field, you had to take account of the fact that while the USAGE and ACTUAL format for the GROUP field are both alphanumeric, the length portion of the USAGE format for the group had to be calculated as the sum of the component lengths, where each integer or single precision field counted as 4 bytes, each double precision field as 8 bytes, and each packed field counted as either 8 or 16 bytes depending on its size.

To avoid the need to calculate these lengths, you can use the GROUP ELEMENTS option, which describes a group as a set of elements without USAGE and ACTUAL formats.

Syntax: How to Describe a GROUP Field as a Set of Elements

```
GROUP=group1, ALIAS=g1alias,ELEMENTS=n1,$
   FIELDNAME=field11, ALIAS=alias11, USAGE=ufmt11, ACTUAL=afmt11, $
   .
   .
   .
   FIELDNAME=field1h, ALIAS=alias1h, USAGE=ufmt1h, ACTUAL=afmt1h, $
GROUP=group2,ALIAS=g2alias,ELEMENTS=n2,$
   FIELDNAME=field21, ALIAS=alias21, USAGE=ufmt21, ACTUAL=afmt21, $
   .
   .
   .
   FIELDNAME=field2k, ALIAS=alias2k, USAGE=ufmt2k, ACTUAL=afmt2k, $
```

where:

group1, group2

Are valid names assigned to a group of fields. The rules for acceptable group names are the same as the rules for acceptable field names.

n1, n2

Are the number of elements (fields and/or groups) that compose the group. If a group is defined within another group, the subgroup (with all of its elements) counts as one element of the parent group.

field11, field2k

Are valid field names.

alias11, alias2k

Are valid alias names.

ufmt11, ufmt2k

Are USAGE formats for each field.

afmt11, afmt2k

Are ACTUAL formats for each field.

Reference: Usage Notes for Group Elements

- ❏ To use the ELEMENTS attribute, the GROUP field declaration should specify only a group name and number of elements.
  - ❏ If a group declaration specifies USAGE and ACTUAL *without* the ELEMENTS attribute, the USAGE and ACTUAL are accepted as specified, even if incorrect.
  - ❏ If a group declaration specifies USAGE and ACTUAL *with* the ELEMENTS attribute, the ELEMENTS attribute takes precedence.
- ❏ Each subgroup counts as one element. Its individual fields and subgroups do not count in the number of elements of the parent group.

Example: Declaring a GROUP With ELEMENTS

In the following Master File, GRP2 consists of two elements, fields FIELDA and FIELDB. GRP1 consists of two elements, GRP2 and field FIELDC. Field FIELDD is not part of a group:

```
FILENAME=XYZ     , SUFFIX=FIX     , $
  SEGMENT=XYZ, SEGTYPE=S2, $
GROUP=GRP1,ALIAS=CCR,ELEMENTS=2,$
   GROUP=GRP2,ALIAS=CC,ELEMENTS=2,$
    FIELDNAME=FIELDA, ALIAS=E01, USAGE=A10, ACTUAL=A10, $
    FIELDNAME=FIELDB, ALIAS=E02, USAGE=A16, ACTUAL=A16, $
    FIELDNAME=FIELDC, ALIAS=E03, USAGE=P27, ACTUAL=A07, $
    FIELDNAME=FIELDD, ALIAS=E04, USAGE=D7, ACTUAL=A07, $
```

The following chart shows the offsets and formats of these fields.

Field Number	Field Name	Offset	USAGE	ACTUAL
1	GRP1	0	A42 - Supports 16 characters for FIELDC (P27)	A33
2	GRP2	0	A26	A26
3	FIELDA	0	A10	A10
4	FIELDB	10	A16	A16
5	FIELDC	26	P27	A7
6	FIELDD	42	D7	A7

Specifying Multilingual Descriptions in a Master File

How to:

Specify Multilingual DESCRIPTION Attributes in a Master File

Activate the Use of a Language

Example:

Using Multilingual Descriptions in a Master File

Reference:

Languages and Language Codes

Usage Notes for Multilingual Metadata

Master Files support descriptions in multiple languages. The description used depends on the value of the LANG parameter and whether a DESC_*ln* attribute is specified in the Master File, where *ln* identifies the language to which the description applies. In order to display these descriptions properly, all of the languages used must be consistent with your NLS configuration.

Syntax: How to Specify Multilingual DESCRIPTION Attributes in a Master File

For a file declaration in a Master File, use the following syntax

```
FILE = filename,
   .
   .
   .
{REMARKS|DESC} = default_desc
DESC_ln = desc_for_ln
   .
   .
   .
```

For a field, use the following syntax

```
FIELDNAME = field, ...
   .
   .
   .
TITLE = default_column_heading
TITLE_ln = column_heading_for_ln
   .
   .
   .
DESC = default_desc
DESC_ln = desc_for_ln
   .
   .
   .
```

where:

field

Is a field in the Master File.

default_column_heading

Is the column heading to use when SET TITLES=ON and either the LANG parameter is set to the default language for the server, or another language is set but the Master File has no corresponding TITLE_ln attribute for that field. This column heading is also used if the *ln* value is invalid.

TITLE_*ln*

Specifies the language to which the associated column heading applies.

column_heading_for_ln

Specifies the text of the column heading for the specified language. That column heading is used when SET TITLES=ON, the LANG parameter is set to a non-default language for FOCUS, and the Master File has a corresponding TITLE_*ln* attribute, where *ln* is the two-digit code for the language specified by the LANG parameter. Valid values for *ln* are the two-letter ISO 639 language code abbreviations.

default_desc

Is the description to use when either the LANG parameter is set to the default language for the server, or another language is set but the Master File has no corresponding DESC_*ln* attribute for that field. This description is also used if an the *ln* value is invalid.

DESC_*ln*

Specifies the language to which the associated description applies.

`desc_for_ln`

Specifies the description text for the specified language. This description is used when the LANG parameter is set to a non-default language for FOCUS and the Master File has a corresponding DESC_*ln* attribute. Valid values for *ln* are the two-letter ISO 639 language code abbreviations.

Reference: Languages and Language Codes

Language Name	Two-Letter Language Code	Three-Letter Language Abbreviation
Arabic	ar	ARB
Baltic	lt	BAL
Chinese - Simplified GB	zh	PRC
Chinese - Traditional Big-5	tw	ROC
Czech	cs	CZE
Danish	da	DAN
Dutch	nl	DUT
English - American	en	AME or ENG
English - UK	uk	UKE
Finnish	fi	FIN
French - Canadian	fc	FRE
French - Standard	fr	FRE
German - Austrian	at	GER
German - Standard	de	GER
Greek	el	GRE
Hebrew	iw	HEW
Italian	it	ITA
Japanese - Shift-JIS(cp942) on ascii cp939 on EBCDIC	ja	JPN

Language Name	Two-Letter Language Code	Three-Letter Language Abbreviation
Japanese - EUC(cp10942) on ascii (UNIX)	je	JPE
Korean	ko	KOR
Norwegian	no	NOR
Polish	pl	POL
Portuguese - Brazilian	br	POR
Portuguese - Portugal	pt	POR
Russian	ru	RUS
Spanish	es	SPA
Swedish	sv	SWE
Thai	th	THA
Turkish	tr	TUR

Syntax: How to Activate the Use of a Language

Issue the following command in a supported profile, on the command line, or in a FOCEXEC:

```
SET LANG = lng
```

or

```
SET LANG = ln
```

In the NLSCGF ERRORS configuration file, issue the following command

```
LANG = lng
```

where:

lng

Is the three-letter abbreviation for the language.

ln

Is the two-letter ISO language code.

Note: If SET LANG is used in a procedure, its value will override the values set in NLSCGF ERRORS or in any profile.

Reference: Usage Notes for Multilingual Metadata

- ❏ To generate the correct characters, all languages used must be on the code page specified at startup.
- ❏ Master Files should be stored using the code page used by FOCUS.
- ❏ Multilingual descriptions are supported with all fields described in the Master File, including DEFINE and COMPUTE fields.
- ❏ The user must create the NLSCFG file. On z/OS, the NLSCFG file must be a member in the concatenation of data sets allocated to DDNAME ERRORS. On z/VM, it must have filetype ERRORS.

Example: Using Multilingual Descriptions in a Master File

The following Master File for the CENTINV data source specifies French descriptions (DESC_FR) and Spanish descriptions (DESC_ES) as well as default descriptions (DESC) for the PROD_NUM and PRODNAME fields:

```
FILE=CENTINV, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=INVINFO, SEGTYPE=S1, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
   DESCRIPTION='Product Number'
   DESC='Product Number',
   DESC_ES='Numero de Producto',
   DESC_FR='Nombre de Produit', $
  FIELD=PRODNAME, ALIAS=PNAME, FORMAT=A30,
   WITHIN=PRODCAT,
   DESCRIPTION='Product Name'
   DESC_FR='Nom de Produit',
   DESC_ES='Nombre de Producto', $
  FIELD=QTY_IN_STOCK, ALIAS=QIS, FORMAT=I7,
   DESCRIPTION='Quantity In Stock', $
  FIELD=PRICE, ALIAS=RETAIL, FORMAT=D10.2,
   TITLE='Price:',
   DESCRIPTION=Price, $
```

Specifying Multilingual TITLE Attributes in a Master File

How to:

Specify Multilingual TITLE Attributes in a Master File

Activate the Use of a Language

Reference:

Usage Notes for Multilingual Titles

Example:

Using Multilingual Titles in a Request

In a Master File, column headings are taken from:

1. A heading specified in the report request using the AS phrase.
2. A TITLE attribute in the Master File, if no AS phrase is specified in the request and SET TITLES=ON.
3. The field name specified in the Master File, if no AS phrase or TITLE attribute is specified, or if SET TITLES=OFF.

Master Files support column headings in multiple languages. The heading used depends on the value of the LANG parameter and whether a TITLE_*ln* attribute is specified in the Master File, where *ln* identifies the language to which the column heading applies.

Syntax: How to Specify Multilingual TITLE Attributes in a Master File

```
FIELDNAME = field, ...
   .
   .
   .
TITLE= default_column_heading
   .
   .
   .
TITLE_ln = column_heading_for_ln
   .
   .
   .
```

where:

field

Is a field in the Master File.

default_column_heading

Is the column heading to use when SET TITLES=ON and either the LANG parameter is set to the default language for FOCUS, or another language is set but the Master File has no corresponding TITLE_*ln* attribute for that field. This column heading is also used if an the *ln* value is invalid.

`TITLE_*ln*`

Specifies the language for which the column heading applies. Valid values for *ln* are the two-letter ISO 639 language code abbreviations:

Language Name	**Two-Letter Language Code**	**Three-Letter Language Abbreviation**
Arabic	ar	ARB
Baltic	lt	BAL
Chinese - Simplified GB	zh	PRC
Chinese - Traditional Big-5	tw	ROC
Czech	cs	CZE
Danish	da	DAN
Dutch	nl	DUT
English - American	en	AME or ENG
English - UK	uk	UKE
Finnish	fi	FIN
French - Canadian	fc	FRE
French - Standard	fr	FRE
German - Austrian	at	GER
German - Standard	de	GER
Greek	el	GRE
Hebrew	he	HEB
Hebrew	iw	HEW
Italian	it	ITA
Japanese - Shift-JIS(cp942) on ascii cp939 on EBCDIC	ja	JPN
Japanese - EUC(cp10942) on ascii (UNIX)	je	JPE
Korean	ko	KOR

Language Name	**Two-Letter Language Code**	**Three-Letter Language Abbreviation**
Norwegian	no	NOR
Polish	pl	POL
Portuguese - Brazilian	br	POR
Portuguese - Portugal	pt	POR
Russian	ru	RUS
Spanish	es	SPA
Swedish	sv	SWE
Thai	th	THA
Turkish	tr	TUR

`column_heading_for_ln`

Is the column heading to use when SET TITLES=ON, the LANG parameter is set to a non-default language for FOCUS, and the Master File has a corresponding TITLE_*ln* attribute, where *ln* is the two-digit code for the language specified by the LANG parameter.

Syntax: How to Activate the Use of a Language

Issue the following command in a supported profile, on the command line, or in a FOCEXEC:

```
SET LANG = lng
```

or

```
SET LANG = ln
```

In the NLSCGF ERRORS configuration file, issue the following command

```
LANG = lng
```

where:

lng

Is the three-letter abbreviation for the language. For a list of supported languages, see *How to Specify Multilingual TITLE Attributes in a Master File*.

ln

Is the two-letter ISO language code. For a list of supported codes, see *How to Specify Multilingual TITLE Attributes in a Master File*.

Note: If SET LANG is used in a procedure, its value will overwrite the values set in NLSCGF ERRORS or in any profile.

Reference: Usage Notes for Multilingual Titles

- To generate the correct characters, all languages used must be on the code page specified at startup. To change the code page, you must stop and restart the server with the new code page.
- Master Files should be stored using the code page used by FOCUS.
- Multilingual titles are supported with all fields described in the Master File, including DEFINE and COMPUTE fields.
- If you issue a HOLD command, only one TITLE attribute is propagated to the HOLD Master File. Its value is the column heading that would have appeared on the report output.
- After it is referenced in a request, the Master File is stored in memory. If you want to produce a report against the same Master File but generate column titles in a different language, you must make sure the Master File is re-read after you issue the SET LANG command and prior to running the request. You can make sure the Master File is re-read by issuing a CHECK FILE command for that Master File.

❑ The user must create the NLSCFG file. On z/OS, the NLSCFG file must be a member in the concatenation of data sets allocated to DDNAME ERRORS. On z/VM, it must have filetype ERRORS.

Example: Using Multilingual Titles in a Request

The following Master File for the CENTINV data source specifies French titles (TITLE_FR) and Spanish titles (TITLE_ES) as well as default titles (TITLE) for the PROD_NUM and PRODNAME fields:

```
FILE=CENTINV, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=INVINFO, SEGTYPE=S1, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
   TITLE='Product,Number:',
   TITLE_FR='Nombre,de Produit:',
   TITLE_ES='Numero,de Producto:',
   DESCRIPTION='Product Number', $
  FIELD=PRODNAME, ALIAS=PNAME, FORMAT=A30,
   WITHIN=PRODCAT,
   TITLE='Product,Name:',
   TITLE_FR='Nom,de Produit:',
   TITLE_ES='Nombre,de Producto:'
   DESCRIPTION='Product Name', $
  FIELD=QTY_IN_STOCK, ALIAS=QIS, FORMAT=I7,
   TITLE='Quantity,In Stock:',
   DESCRIPTION='Quantity In Stock', $
  FIELD=PRICE, ALIAS=RETAIL, FORMAT=D10.2,
   TITLE='Price:',
   DESCRIPTION=Price, $
```

The default language is English and, by default, SET TITLES=ON. Therefore, the following request uses the TITLE attributes to produce column headings that are all in English:

```
TABLE FILE CENTINV
PRINT PROD_NUM PRODNAME PRICE
WHERE PRICE LT 200
END
```

The output is:

```
Product  Product
Number:  Name:                                 Price:
-------  -------                               ------
1004     2 Hd VCR LCD Menu                     179.00
1008     DVD Upgrade Unit for Cent. VCR        199.00
1026     AR3 35MM Camera 10 X                  129.00
1028     AR2 35MM Camera 8 X                   109.00
1030     QX Portable CD Player                 169.00
1032     R5 Micro Digital Tape Recorder         89.00
```

Now, issue the following command to set the language to Spanish:

```
SET LANG = SPA
```

Issue the CHECK FILE CENTINV command to re-read the Master File, and rerun the request.

The output now displays column headings from the TITLE_ES attributes where they exist (Product Number and Product Name). Where no Spanish title is specified (the Price field), the column heading in the TITLE attribute appears:

```
Numero        Nombre
de Producto:  de Producto:                        Price:
------------  ------------                        ------
1004          2 Hd VCR LCD Menu                   179.00
1008          DVD Upgrade Unit for Cent. VCR      199.00
1026          AR3 35MM Camera 10 X                129.00
1028          AR2 35MM Camera 8 X                 109.00
1030          QX Portable CD Player               169.00
1032          R5 Micro Digital Tape Recorder       89.00
```

Using Date System Amper Variables in Master File DEFINEs

Reference:

Messages for Date System Amper Variables in Master File DEFINEs

Example:

Using the Date Variable &DATE in a Master File DEFINE

Using the Date Variable &YYMD in a Master File DEFINE

Master File DEFINE fields can use Dialogue Manager system date variables to capture the system date each time the Master File is parsed for use in a request.

The format of the returned value for each date variable is the format indicated in the variable name. For example, &DATEYYMD returns a date value with format YYMD. The exceptions are &DATE and &TOD, which return alphanumeric values and must be assigned to a field with an alphanumeric format. The variable names &DATE and &TOD must also be enclosed in single quotation marks in the DEFINE expression.

The variables supported for use in Master File DEFINEs are:

- &DATE
- &TOD
- &DATEMDY
- &DATEDMY
- &DATEYMD
- &DATEMDYY
- &DATEDMYY
- &DATEYYMD
- &DMY
- &YMD
- &MDY
- &YYMD
- &MDYY
- &DMYY

Note that all other reserved amper variables are not supported in Master Files.

Example: Using the Date Variable &DATE in a Master File DEFINE

The following version of the EMPLOYEE Master File has the DEFINE field named TDATE added to it. TDATE has format A12 and retrieves the value of &DATE, which returns an alphanumeric value and must be enclosed in single quotation marks:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,        ALIAS=EID,     FORMAT=A9,       $
 FIELDNAME=LAST_NAME,     ALIAS=LN,      FORMAT=A15,      $
 FIELDNAME=FIRST_NAME,    ALIAS=FN,      FORMAT=A10,      $
 FIELDNAME=HIRE_DATE,     ALIAS=HDT,     FORMAT=I6YMD,    $
 FIELDNAME=DEPARTMENT,    ALIAS=DPT,     FORMAT=A10,      $
 FIELDNAME=CURR_SAL,      ALIAS=CSAL,    FORMAT=D12.2M,   $
 FIELDNAME=CURR_JOBCODE, ALIAS=CJC,     FORMAT=A3,       $
 FIELDNAME=ED_HRS,        ALIAS=OJT,     FORMAT=F6.2,     $
DEFINE TDATE/A12   ='&DATE';, $
   .
   .
   .
```

The following request displays the value of TDATE:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE TDATE AS 'TODAY''S,DATE'
WHERE LAST_NAME EQ 'BANNING'
END
```

The output is:

```
                                                TODAY'S
LAST_NAME         FIRST_NAME  HIRE_DATE  DATE
---------         ----------  ---------  -------
BANNING           JOHN         82/08/01  05/11/04
```

Example: Using the Date Variable &YYMD in a Master File DEFINE

The following version of the EMPLOYEE Master File has the DEFINE field named TDATE added to it. TDATE has format YYMD and retrieves the value of &YYMD:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,        ALIAS=EID,     FORMAT=A9,       $
 FIELDNAME=LAST_NAME,     ALIAS=LN,      FORMAT=A15,      $
 FIELDNAME=FIRST_NAME,    ALIAS=FN,      FORMAT=A10,      $
 FIELDNAME=HIRE_DATE,     ALIAS=HDT,     FORMAT=I6YMD,    $
 FIELDNAME=DEPARTMENT,    ALIAS=DPT,     FORMAT=A10,      $
 FIELDNAME=CURR_SAL,      ALIAS=CSAL,    FORMAT=D12.2M,   $
 FIELDNAME=CURR_JOBCODE,  ALIAS=CJC,     FORMAT=A3,       $
 FIELDNAME=ED_HRS,        ALIAS=OJT,     FORMAT=F6.2,     $
DEFINE TDATE/YYMD    = &YYMD ;, $
   .
   .
   .
```

The following request displays the value of TDATE:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE TDATE AS 'TODAY''S,DATE'
WHERE LAST_NAME EQ 'BANNING'
END
```

The output is:

```
                                                TODAY'S
LAST_NAME         FIRST_NAME  HIRE_DATE  DATE
---------         ----------  ---------  -------
BANNING           JOHN         82/08/01  2004/05/11
```

Reference: Messages for Date System Amper Variables in Master File DEFINEs

The following message appears if an attempt is made to use an unsupported amper variable in a Master File DEFINE:

```
(FOC104) DEFINE IN MASTER REFERS TO A FIELD OUTSIDE ITS SCOPE: var
```

CREATE FILE DROP

How to:

Issue the CREATE FILE Command With the DROP Option

Reference:

Usage Notes for CREATE FILE DROP

Example:

Creating a FOCUS Data Source That Already Exists

Creating an Oracle Table That Already Exists

The CREATE FILE command creates a data source that conforms to an existing Master File. You can use this command to create a FOCUS, XFOCUS, or relational data source.

If you issue the CREATE FILE command when the data source already exists, the following message appears for a FOCUS or XFOCUS data source:

```
(FOC441) WARNING. THE FILE EXISTS ALREADY. CREATE WILL WRITE OVER IT
```

For a relational data source, the following messages appear, followed by messages from the Relational engine indicating that the table cannot be created and then by a FOC1414 message:

```
(FOC1400) SQLCODE IS 955 (HEX: 000003BB)
(FOC1421) TABLE EXISTS ALREADY. DROP IT OR USE ANOTHER TABLENAME
```

The DROP option on the CREATE FILE command prevents the display of the messages and creates the data source, dropping the existing table first if necessary and re-parsing the Master File if it changed.

Syntax: How to Issue the CREATE FILE Command With the DROP Option

```
CREATE FILE filename DROP
```

where:

filename

Is the Master File name for a FOCUS, XFOCUS, or relational data source.

Example: Creating a FOCUS Data Source That Already Exists

The following CREATE FILE command creates the EMPLOYEE data source, which already exists:

```
create file employee
(FOC441) WARNING. THE FILE EXISTS ALREADY. CREATE WILL WRITE OVER IT

REPLY :
```

If you reply NO, the file is not created. If you reply anything else, the file is created.

The following CREATE FILE command creates the EMPLOYEE data source, which already exists without generating the FOC441 warning or requesting a reply.

```
create file employee drop
 NEW FILE EMPLOYEE  FOCUS   A1 ON 05/12/2004 AT 16.09.14
```

Example: Creating an Oracle Table That Already Exists

The Oracle table name EMPINFO exists therefore, the following CREATE FILE command generates an error and the new version of the table is not created:

```
create file empinfo
(FOC1400) SQLCODE IS 955 (HEX: 000003BB)
(FOC1421) TABLE EXISTS ALREADY. DROP IT OR USE ANOTHER TABLENAME
  : ORA-00955: name is already used by an existing object
  : Erroneous token: EMPINFO
(FOC1414) EXECUTE IMMEDIATE ERROR.
```

The following CREATE FILE command drops the existing table and creates a new version:

```
create file empinfo drop
>
```

The SQLDI trace shows that the original version of the table is dropped and then the new version is created:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = SQLDI//CLIENT
create file empinfo drop

**** orafoc: gfun: 3, fun: 2, cnum: 0
**** orafoc: gfun: 1, fun: 6, cnum: 0
**** Local server assigned for operation
**** OOPEN call
**** OOPEN return
**** acda=008DC0C8,ocnum=1,retcode=0
**** OPARSE call
**** l of sql cmd  20
SQL:  DROP TABLE EMPINFO
**** lngflg value  1
**** OPARSE return
**** acda=008DC0C8,ocnum=1,retcode=0
**** OEXEC call:
**** OEXEC return
**** acda=008DC0C8,ocnum=1,retcode=0
**** orafoc: gfun: 1, fun: 6, cnum: 0
**** Local server assigned for operation
**** OPARSE call
**** l of sql cmd  259
SQL:  CREATE TABLE EMPINFO( "EID" VARCHAR2 (9) NOT NULL ,"LN" VARCHAR2
SQL: (15) NOT NULL ,"FN" VARCHAR2 (10) NOT NULL ,"HDT" DATE  NOT NULL ,"
SQL: DPT" VARCHAR2 (10),"CSAL" DECIMAL(7, 2) NOT NULL ,"CJC" VARCHAR2 (3
SQL: ) NOT NULL ,"OJT" FLOAT ,"BONUS_PLAN" INTEGER  NOT NULL )
**** lngflg value  1
**** OPARSE return
**** acda=008DC0C8,ocnum=1,retcode=0
**** OEXEC call:
**** OEXEC return
```

```
**** acda=008DC0C8,ocnum=1,retcode=0
**** orafoc: gfun: 1, fun: 6, cnum: 0
**** Local server assigned for operation
**** OPARSE call
**** 1 of sql cmd  57
SQL:  CREATE  UNIQUE INDEX EMPINFOIX ON EMPINFO ("EID"  ASC)
**** lngflg value  1
**** OPARSE return
**** acda=008DC0C8,ocnum=1,retcode=0
**** OEXEC call:
**** OEXEC return
**** acda=008DC0C8,ocnum=1,retcode=0
**** orafoc: gfun: 1, fun: 5, cnum: 0
**** OCLOSE call

**** OCLOSE return
**** acda=008DC0C8,ocnum=0,retcode=0
**** OCOM call
**** OCOM return
**** orafoc: gfun: 3, fun: 3, cnum: 0
```

Reference: Usage Notes for CREATE FILE DROP

If you issue the CREATE FILE *filename* DROP command for a FOCUS or XFOCUS data source that has an external index or MDI, you must REBUILD the index after creating the data source.

COMPUTE in a Master File

COMPUTE commands can be included in Master Files and referenced in subsequent TABLE requests, enabling you to build expressions once and use them in multiple requests.

Syntax: How to Include a COMPUTE Command in a Master File

```
COMPUTE fieldname/fmt=expression;
```

where:

fieldname

Is name of the calculated field.

fmt

Is the format and length of the calculated field.

expression

Is the formula for calculating the value of the field.

Reference: Usage Notes for COMPUTE in a Master File

In all instances, COMPUTEs in the Master File have the same functionality and limitations as temporary COMPUTEs. Specifically, fields computed in the Master File must follow these rules:

- They cannot be used in JOIN, DEFINE, or ACROSS phrases, or with prefix operators.
- When used as selection criteria, syntax is either IF TOTAL field or WHERE TOTAL field.
- When used as sort fields, syntax is BY TOTAL COMPUTE field.
- To insert a calculated value into a heading or footing, you must reference it prior to the HEADING or FOOTING command.

Example: Coding a COMPUTE in the Master File and Accessing the Computed Value

Use standard COMPUTE syntax to add a calculated value to your Master File. You can then access the calculated value by referencing the computed fieldname in subsequent TABLE requests. When used as a verb object, as in the following example, the syntax is SUM (or PRINT) COMPUTE field.

The following is the SALESTES Master File (the SALES FILE modified with an embedded COMPUTE):

```
FILENAME=SALESTES, SUFFIX=FOC,
SEGNAME=STOR_SEG, SEGTYPE=S1,
   FIELDNAME=STORE_CODE,  ALIAS=SNO,  FORMAT=A3,   $
   FIELDNAME=CITY,        ALIAS=CTY,  FORMAT=A15,  $
   FIELDNAME=AREA,        ALIAS=LOC,  FORMAT=A1,   $

SEGNAME=DATE_SEG, PARENT=STOR_SEG, SEGTYPE=SH1,
   FIELDNAME=DATE,        ALIAS=DTE,  FORMAT=A4MD, $

SEGNAME=PRODUCT, PARENT=DATE_SEG, SEGTYPE=S1,
   FIELDNAME=PROD_CODE,     ALIAS=PCODE,   FORMAT=A3,    FIELDTYPE=I, $
   FIELDNAME=UNIT_SOLD,     ALIAS=SOLD,    FORMAT=I5,    $
   FIELDNAME=RETAIL_PRICE,  ALIAS=RP,      FORMAT=D5.2M, $
   FIELDNAME=DELIVER_AMT,   ALIAS=SHIP,    FORMAT=I5,    $
   FIELDNAME=OPENING_AMT,   ALIAS=INV,     FORMAT=I5,    $
   FIELDNAME=RETURNS,       ALIAS=RTN,     FORMAT=I3,    MISSING=ON, $
   FIELDNAME=DAMAGED,       ALIAS=BAD,     FORMAT=I3,    MISSING=ON, $

   COMPUTE REVENUE/D12.2M=UNIT_SOLD*RETAIL_PRICE;
```

In the TABLE request, computed field, REVENUE, is a verb object of SUM.

```
TABLE FILE SALESTES
HEADING CENTER
"NEW YORK PROFIT REPORT"
" "
SUM UNIT_SOLD AS 'UNITS,SOLD' RETAIL_PRICE AS 'RETAIL_PRICE'
COMPUTE REVENUE;
BY PROD_CODE AS 'PROD,CODE'
WHERE CITY EQ 'NEW YORK'
END
```

The output is:

```
          NEW YORK PROFIT REPORT

 PROD  UNITS
 CODE  SOLD   RETAIL_PRICE         REVENUE
 ----  ----   ------------         -------
 B10     30            $.85         $25.50
 B17     20           $1.89         $37.80
 B20     15           $1.99         $29.85
 C17     12           $2.09         $25.08
 D12     20           $2.09         $41.80
 E1      30            $.89         $26.70
 E3      35           $1.09         $38.15
```

Comma-delimited Access File

How to:

Create an Access File

Reference:

Access File Attributes for Comma-Delimited Access Files\

Example:

Access File for the VIDEOTR2 Data Source

Starting in Version 7.6.5, a comma-delimited Access File has been implemented for FOCUS and XFOCUS data sources.

The Access File for a FOCUS or XFOCUS data source is needed to implement intelligent partitioning or to access a Multi-dimensional index (MDI). You can also use it to specify the data set names for location segments.

In prior releases, the Access File for a FOCUS data source was space delimited and could describe the files associated with several Master Files. The name of the Access File was arbitrary and was specified with the ACCESS = attribute in the Master File.

The new syntax is comma-delimited, similar to the Master File syntax. It can describe the files needed for accessing only one Master File. The name of the Access File must be the same as the Master File name, and it must be specified with the ACCESS = attribute in the Master File.

Both types of syntax are supported in the FOCUS 7.6 track, but starting with FOCUS 7.7, only the comma-delimited syntax will be supported.

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 contains an ACCESS= attribute that references an Access File, the Access File is read and used to locate the correct data sources. With the comma-delimited syntax, the Access File must have the same name as the Master File. With the space-delimited syntax, the Access File can have any name. If there is no Access File with the same name as the ACCESS= attribute in the Master File, the request is processed with the Master File alone.

An Access File is required to take advantage of intelligent partitioning. Intelligent partitioning places records containing specific data values in separate physical partitions and uses the Access File to describe the values in the records in each partition. With this information, data access is optimized by retrieving only those partitions whose values are consistent with the selection criteria in the request.

Note: On z/OS, the Access File must be a member of a data set concatenated in the allocation for ddname ACCESS. On z/VM, the Access File must have the file type ACCESS. The Access File has the same DCB attributes as the Master File. On UNIX and Windows, the Access File has the extension .acx.

Reference: Access File Attributes for Comma-Delimited Access Files

Each comma-delimited Access File describes the files and MDIs for one Master File, and that Master File must have the same file name as the Access File.

All attribute/value pairs are separated by an equal sign (=), and each pair in a declaration is delimited with a comma (,). Each declaration is terminated with the comma dollar sign (,$).

1. Each Access File starts with a declaration that names its corresponding Master File.
2. Next comes the DATA declaration that describes the location of the physical file. If the file is partitioned, it has multiple DATA declarations.

   If the file is *intelligently* partitioned so that an expression describes which data values reside in each partition, the DATA declaration has a WHERE phrase that specifies this expression.
3. If the data source has LOCATION segments the LOCATION declaration names a location segment. Its corresponding LOCATIONDATA declaration points to the physical LOCATION file.
4. If the data source has an MDI, the Access File has an MDI declaration that names the MDI and its target segment, followed by declarations that name the dimensions of the MDI, followed by the MDIDATA declaration that points to the physical MDI file. If the MDI is partitioned, there are multiple MDIDATA declarations for the MDI.

Syntax: How to Create an Access File

Master File declaration:

```
MASTER=mastername,$
```

where:

mastername

Indicates the name of the Master File with which this Access File is associated. It is the same value included in the Master File ACCESS=*filename* attribute, used to indicate both the existence of the Access File and its name.

```
DATA=file_specification,
   [WHERE= expression; ,]$
[DATA=file_specification,
   [WHERE= expression; ,]$ ...]
```

where:

file_specification

Points to the file location. This is a complete file specification. There are can be up to 250 DATA declarations (partitions) in a single Access File. With XFOCUS data sources, this supports creation of a 4 Terabtye database. Using FOCUS data sources, a 500 GB database can be constructed.

The WHERE clause is the basis of the Intelligent Partitioning feature. The expression is terminated with the semi-colon and the entire declaration with the comma/dollar sign. WHERE expressions of the following type are supported:

```
WHERE = field  operator value1 [ OR  value2...]; ,$
```

```
WHERE = field FROM value1 TO value2 [AND FROM value3 TO value4];,$
```

Expressions can be combined with the AND operator.

Location File declarations:

```
LOCATION=location_segment_name,$
  LOCATIONDATA=location_segment_file_specification,$
```

where:

location_segment_name

Is the name of the segment stored in the location file.

location_segment_file_specification

Is the full file specification for the physical file the segment is located in.

MDI declarations:

```
MDI=mdiname, TARGET_OF = segname,$
   DIM = [filename.]fieldname [, MAXVALUES = n] ,$
   [DIM = [filename.]fieldname [, MAXVALUES = n] ,$ ...]
 MDIDATA=mdi_file_specification,$
 [MDIDATA=mdi_file_specification,$ ...]
```

where:

mdiname

Is the name of the MDI.

segname

Is the name of the target segment

filename

Is the name of the file where an MDI dimension resides.

fieldname

Is the name of a field that is a dimension of the MDI.

n

Is the number of distinct values in the dimension. When the MDI is created, the actual dimension value will be converted to an integer of length 1, 2, or 4 bytes, and this number will be stored in the index leaf.

mdi_file_specification

Is the fully-qualified specification of the physical MDI file. If the MDI is partitioned, it is the specification for one partition of the MDI. An MDI can have up to 250 MDIDATA declarations (partitions). An Access File can have an unlimited number of MDIs.

Example: Access File for the VIDEOTR2 Data Source

VIDEOTR2 is an intelligently partitioned FOCUS data source. The Master File has an ACCESS=VIDEOTR2 attribute:

```
FILENAME=VIDEOTR2,  SUFFIX=FOC,    ACCESS=VIDEOTR2
SEGNAME=CUST,       SEGTYPE=S1
 FIELDNAME=CUSTID,       ALIAS=CIN,          FORMAT=A4,       $
 FIELDNAME=LASTNAME,     ALIAS=LN,           FORMAT=A15,      $
 FIELDNAME=FIRSTNAME,    ALIAS=FN,           FORMAT=A10,      $
 FIELDNAME=EXPDATE,      ALIAS=EXDAT,        FORMAT=YMD,      $
 FIELDNAME=PHONE,        ALIAS=TEL,          FORMAT=A10,      $
 FIELDNAME=STREET,       ALIAS=STR,          FORMAT=A20,      $
 FIELDNAME=CITY,         ALIAS=CITY,         FORMAT=A20,      $
 FIELDNAME=STATE,        ALIAS=PROV,         FORMAT=A4,       $
 FIELDNAME=ZIP,          ALIAS=POSTAL_CODE,  FORMAT=A9,       $
 FIELDNAME=EMAIL,        ALIAS=EMAIL,        FORMAT=A18,      $
SEGNAME=TRANSDAT, SEGTYPE=SH1,  PARENT=CUST
 FIELDNAME=TRANSDATE,    ALIAS=OUTDATE,   FORMAT=HYYMDI,
   MISSING=ON, $
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 following shows the Access File, named VIDEOTR2, on z/OS:

```
MASTER=VIDEOTR2 ,$
  DATA=USER1.VIDPART1.FOCUS,
    WHERE=DATE EQ 1991;,$

  DATA=USER1.VIDPART2.FOCUS,
    WHERE=DATE FROM 1996 TO 1998; ,$

  DATA=USER1.VIDPART3.FOCUS,
   WHERE=DATE FROM 1999 TO 2000;,$
```

The following shows the Access File, named VIDEOTR2, on CMS:

```
MASTER=VIDEOTR2 ,$
  DATA='VIDPART1 FOCUS A',
    WHERE=DATE EQ 1991;,$

  DATA='VIDPART2 FOCUS A',
    WHERE=DATE FROM 1996 TO 1998; ,$

  DATA='VIDPART3 FOCUS A',
   WHERE=DATE FROM 1999 TO 2000;,$
```

1022 Partitions

Reference:

Usage Notes for 1022 Partitions

Starting with Version 7.6.5, FOCUS and XFOCUS data sources can consist of up to 1022 physical files. The number of physical files associated with one FOCUS or XFOCUS data source is the sum of its partitions and LOCATION files. This sum must be less than or equal to 1022. FOCUS or XFOCUS data sources can grow in size over time, and can be re-partitioned based on the requirements of the application.

In order to concatenate multiple partitions, you must have a FOCUS Access File or a USE command. Both of these now support up to 1022 files for a data source. For complete information about partitioning a data source and creating an Access File, see the *Describing Data* manual.

Reference: Usage Notes for 1022 Partitions

The following guidelines apply:

- ❏ Concatenation of multiple partitions is supported for reporting only.
- ❏ The total number of files in a join structure can be up to 1022 counting all host and cross-referenced partitions.
- ❏ There is no increase to the number of external index or MDI partitions. They are limited to 240 partitions.

6 MODIFY Enhancements

This chapter describes enhancements to the MODIFY facility.

For information about compiling MODIFY expressions using native arithmetic, see *Compiling MODIFY Expressions Using Native Arithmetic* in Chapter 1, *Performance Enhancements*.

Topics:

Loading Fixed Format Sequential Files Using MODIFY

How to:

Load Data Into Fixed Format Sequential Files Using MODIFY

Reference:

Usage Notes for MODIFY of Fixed Format Sequential Data Sources

Example:

Loading a Fixed Format Sequential Data Source Using MODIFY

Reporting on a Fixed Format File That is Out of Sequence

You can use the MODIFY command to load data into a single segment fixed format sequential data source. This enables you to created fixed format files with specific data types that are difficult to create using the HOLD FORMAT INTERNAL command.

The data is loaded in the order in which it is input. Update and delete operations are not supported.

Syntax: How to Load Data Into Fixed Format Sequential Files Using MODIFY

```
MODIFY FILE filename
readcmd
[COMPUTE field[/format1] = expression1;]
[VALIDATE testfld[/format2] = expression2;]
modify_select_and_load_cmds
DATA [ON ddname] [VIA CRTFORM]
END
```

where:

filename

Is the name of the fixed format sequential file to be loaded.

readcmd

Can be any supported MODIFY command for describing and reading incoming data such as CRTFORM, PROMPT, FREEFORM, or FIXFORM.

field

Can be an existing field in the data source, a temporary field, or an input field that requires calculations in order to be used in your MODIFY processing.

format1

Is the format you assign to a temporary field. Omit the format for existing fields.

testfld

Is a temporary field that will be used to accept or reject a transaction based on a calculation. If the value of the field is zero, the transaction will be rejected.

format2

Is the format you assign to the field. The format type must be numeric (I, F, D, or P. Specify the format only if you will use the field elsewhere in the request.

expression1, expression2

Are expressions supported in MODIFY.

modify_select_and_load_cmds

Can be any sequence of MODIFY commands that select transactions, such as MATCH and NEXT, as well as INCLUDE commands that load records into the data source.

ddname

Specifies the logical name of a data source that contains the transaction records.

For complete information about MODIFY facilities and commands, see your MODIFY documentation.

Reference: Usage Notes for MODIFY of Fixed Format Sequential Data Sources

- ❑ MATCH only positions to the first data source record that matches the first transaction record. To move through the file after a MATCH, use NEXT.
- ❑ The SEGTYPE is ignored when matching. Therefore, the recommended SEGTYPE is S0.
- ❑ Any NOMATCH or INCLUDE command that is executed, positions you at the end of the file.
- ❑ REPOSITION is not supported.
- ❑ Multi-segment fixed format sequential files are not supported.
- ❑ Data is loaded in the sequence in which it is input.
- ❑ If the data source already exists, new records are added at the end.
- ❑ To report on a fixed format file that is out of sequence but has a SEGTYPE that specifies one or more key fields (for example, S1or S2), issue the SET FIXRETRIEVE=OFF command. This setting causes the entire file to be searched for records that satisfy a screening condition in the report request.

Example: Loading a Fixed Format Sequential Data Source Using MODIFY

The following Master File describes a fixed format data source called CENTFIX. This Master File is based on the Master File for the FOCUS data source named CENTINV, with the SUFFIX changed to FIX, the SEGTYPE changed to S0, and the DEFINE fields removed:

```
FILE=CENTFIX, SUFFIX=FIX, FDFC=19, FYRT=00
 SEGNAME=INVINFO, SEGTYPE=S0, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
   TITLE='Product,Number:',
   DESCRIPTION='Product Number', $
  FIELD=PRODNAME, ALIAS=PNAME, FORMAT=A30,
   WITHIN=PRODCAT,
   TITLE='Product,Name:',
   DESCRIPTION='Product Name', $
  FIELD=QTY_IN_STOCK, ALIAS=QIS, FORMAT=I7,
   TITLE='Quantity,In Stock:',
   DESCRIPTION='Quantity In Stock', $
  FIELD=PRICE, ALIAS=RETAIL, FORMAT=D10.2,
   TITLE='Price:',
   DESCRIPTION=Price, $
  FIELD=COST, ALIAS=OUR_COST, FORMAT=D10.2,
   TITLE='Our,Cost:',
   DESCRIPTION='Our Cost:', $
```

The following procedure loads the fixed format sequential file named CENTFIX. The input data is the same as the data used to load the FOCUS data source CENTINV. However, the record for PROD_NUM '1022' has been moved out of sequence, in front of the record for PROD_NUM '1020':

```
CMS FILEDEF CENTFIX DISK CENTFIX DATA A
CREATE FILE CENTFIX
MODIFY FILE CENTFIX
FIXFORM PROD_NUM/4 PRODNAME/30 QTY_IN_STOCK/7 PRICE/12 COST/12
CHECK OFF
MATCH PROD_NUM
ON MATCH REJECT
ON NOMATCH INCLUDE
DATA
10042 Hd VCR LCD Menu                   43068      179.00      129.00
1006Combo Player - 4 Hd VCR + DVD   13527      399.00      289.00
1008DVD Upgrade Unit for Cent. VCR    199      199.00      139.00
1010750SL Digital Camcorder 300 X   10758      999.00      750.00
1012650DL Digital Camcorder 150 X    2972      899.00      710.00
1014340SX Digital Camera 65K P        990      249.00      199.00
1016330DX Digital Camera 1024K P    12707      279.00      199.00
1018250 8MM Camcorder 40 X          60073      399.00      320.00
1022120 VHS-C Camcorder 40 X         2300      399.00      259.00
1020150 8MM Camcorder 20 X           5961      319.00      240.00
1024110 VHS-C Camcorder 20 X         4000      349.00      249.00
1026AR3 35MM Camera 10 X            12444      129.00       95.00
1028AR2 35MM Camera 8 X             11499      109.00       79.00
1030QX Portable CD Player           22000      169.00       99.00
1032R5 Micro Digital Tape Recorder   1990       89.00       69.00
1034ZT Digital PDA - Commercial     21000      499.00      349.00
1036ZC Digital PDA - Standard       33000      299.00      249.00
END
```

The following messages indicate that the data was loaded:

```
 CENTFIX FIX       ON 07/14/2006 AT 10.17.05
(FOC1291) RECORDS AFFECTED DURING CURRENT REQUEST  : 17/INSERT
 TRANSACTIONS:         TOTAL =    17  ACCEPTED=    17  REJECTED=     0
 SEGMENTS:             INPUT =    17  UPDATED =     0  DELETED =     0
```

The following request reports from the CENTFIX fixed format file:

```
CMS FILEDEF CENTFIX DISK CENTFIX DATA A
TABLE FILE CENTFIX
SUM PRODNAME PRICE COST
BY PROD_NUM
END
```

The output is:

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

Example: Reporting on a Fixed Format File That is Out of Sequence

The CENTFIX data source was loaded with the record for PROD_NUM '1022' before the record for PROD_NUM '1020'. This does not cause a problem for reporting if the SEGTYPE is S0.

However, if the SEGTYPE for CENTFIX is S1, PROD_NUM is considered the key field, and is a candidate for keyed retrieval. Keyed retrieval assumes that the records are in order of the key field. If a value higher than the requested value is found in the data source, it is assumed that no record exists with the requested value, and retrieval halts.

Note: The SEGTYPE would be S1 if CENTFIX had been created as a HOLD file based on the CENTINV data source, and if the Master File had not been manually edited to change the SEGTYPE value.

Consider the following request that prints the record in which PROD_NUM equals '1020' and in which FIXRETRIEVE is ON (the default value):

```
TABLE FILE CENTFIX
PRINT PROD_NUM PRODNAME PRICE COST
WHERE PROD_NUM EQ '1020'
ON TABLE SET FIXRETRIEVE ON
END
```

The equality test does not retrieve the record with PROD_NUM='1020' because the keyed retrieval assumes the records are in order of the key field and stops searching when the record for PROD_NUM '1022' is found:

```
NUMBER OF RECORDS IN TABLE=        0  LINES=      0
```

Running the request with FIXRETRIEVE OFF retrieves the requested record even though it is out of sequence:

```
TABLE FILE CENTFIX
PRINT PROD_NUM PRODNAME PRICE COST
WHERE PROD_NUM EQ '1020'
ON TABLE SET FIXRETRIEVE OFF
END
```

The output is:

```
Product  Product                                        Our
Number:  Name:                             Price:       Cost:
-------  -------                           ------       -----
1020     150 8MM Camcorder 20 X            319.00      240.00
```

MODIFY FIXFORM Support for Multiple Text Fields

How to:

Load Multiple Text Fields Using a FIXFORM Field List

Load Multiple Text Fields Using a HOLD File With FIXFORM

Example:

Loading Multiple Text Fields Using FIXFORM FROM HOLD

MODIFY FIXFORM can load multiple text fields into a data source. The data source must have a Write data adapter that supports MODIFY, and the data source must support text fields. In the case of Relational data sources, long varchar fields are mapped as text fields in the Master File. Support for this feature with a relational data adapter is dependent on whether the specific relational engine supports multiple long varchar columns.

The text fields must be the last fields in the FIXFORM field list and *may not* be conditional transaction fields. In the file, each text field must be terminated by a %$ character combination on a line by itself.

If the data to be loaded is from a HOLD file, the text fields must also be the last fields in the HOLD file. However, the text fields can be loaded anywhere in the receiving data source.

Syntax: How to Load Multiple Text Fields Using a FIXFORM Field List

```
FIXFORM field1/fmt1 ... fieldn/fmtn txtfld1/TX ... txtfldn/TX
```

where:

`field1 ... fieldn`

Are non-text fields.

`fmt1 ... fmtn`

Are the format specifications for the non-text fields.

`txtfld1 ... txtfldn`

Are text fields.

`TX`

Is the format specification for the text fields.

Syntax: How to Load Multiple Text Fields Using a HOLD File With FIXFORM

```
FIXFORM FROM ddname
```

where:

`ddname`

Is the AS name used to create the HOLD file. If no AS phrase was specified in the HOLD command, the ddname is HOLD.

Example: Loading Multiple Text Fields Using FIXFORM FROM HOLD

This example uses the COURSES data source, which contains a text field, to create a HOLD file with three text fields. It then uses this HOLD file to load data into the TEXT3 data source.

The following request creates the HOLD file:

```
SET ASNAMES=ON
TABLE FILE COURSES
PRINT COURSE_CODE DESCRIPTION AS 'DESCRIPTION1'
DESCRIPTION/TX25 AS 'DESCRIPTION2'
DESCRIPTION/TX100 AS 'DESCRIPTION3'
ON TABLE HOLD FORMAT ALPHA
END
```

The following is the Master File for the TEXT3 data source:

```
FILE=TEXT3          ,SUFFIX=FOC
SEGNAME=SEG1,SEGTYPE=S1
FIELDNAME   =COURSE_CODE        ,CCODE       ,A6             ,$
FIELDNAME   =DESCRIPTION1       ,DESC1       ,TX50           ,$
FIELDNAME   =DESCRIPTION2       ,DESC2       ,TX25           ,$
FIELDNAME   =DESCRIPTION3       ,DESC3       ,TX100          ,$
```

The following procedure loads the TEXT3 data source:

```
CREATE FILE TEXT3
MODIFY FILE TEXT3
FIXFORM FROM CRSEHOLD
DATA ON CRSEHOLD
END
```

The following messages indicate that the data was loaded:

```
 NEW FILE TEXT3   FOCUS   A1 ON 06/04/2004 AT 16.02.18
>
 TEXT3   FOCUS   A1 ON 06/04/2004 AT 16.02.18
 TRANSACTIONS:         TOTAL =     3  ACCEPTED=     3  REJECTED=     0
 SEGMENTS:             INPUT =     3  UPDATED =     0  DELETED =     0
>
```

The following request prints the second text field from the data source:

```
TABLE FILE TEXT3
PRINT DESCRIPTION2
BY COURSE_CODE
ON COURSE_CODE SKIP-LINE
END
```

The output is:

```
COURSE_CODE  DESCRIPTION2
-----------  ------------

101          This course provides the
             DP professional with the
             skills needed to create,
             maintain, and report from
             FOCUS databases.

200          Anyone responsible for
             designing FOCUS databases
             will benefit from this
             course, which provides
             the skills needed to
             design large, complex
             databases and tune
             existing ones.

201          This is a course in FOCUS
             efficiencies.
```

Controlling Whether FIXFORM Input Fields Are Conditional

How to:

Control Whether FIXFORM Input Fields Are Conditional

Reference:

Usage Notes for SET FIXFRMINPUT

Example:

Controlling Whether FIXFORM Transaction Fields Are Conditional

In MODIFY, by default, FIXFORM FROM *mastername* treats all transaction data as conditional, meaning that space-filled fields are considered not present, and as such cannot be updated or used in updates.

The SET FIXFRMINPUT command enables you to specify how to handle FIXFORM input fields as either conditional (field/format C) or non-conditional fields. Thus, spaces in a transaction field can be used for updating database fields.

Syntax: How to Control Whether FIXFORM Input Fields Are Conditional

```
SET FIXFRMINPUT = {COND|NONCOND}
```

where:

COND

Treats all transaction fields generated by FIXFORM FROM *mastername* as conditional (format C) fields. COND is the default value.

NONCOND

Treats all transaction fields as present in the transaction, and their contents are treated as real values.

Note that if you have not changed the value of the FIXFRMINPUT parameter and you query its value, the value displays as DEFAULT.

Reference: Usage Notes for SET FIXFRMINPUT

- The FIXFRMINPUT setting does not affect a FIXFORM command that does not have a FROM phrase.
- If you run a compiled MODIFY, its behavior reflects the FIXFRMINPUT setting at the time it was compiled, even if a different setting is in effect at run time.

Example: Controlling Whether FIXFORM Transaction Fields Are Conditional

The following procedure establishes a transaction file, defining LN1 in HOLD file TRANS to be blank for PIN 000000040.

```
SET ASNAMES = ON
DEFINE FILE EMPDATA
LN1/A15 = IF PIN EQ '000000040' THEN '' ELSE LN;
END
TABLE FILE EMPDATA
PRINT PIN LN1 AS LN
IF PIN FROM '000000010' TO '000000100'
ON TABLE HOLD AS TRANS
END
```

The following procedure, sets the FIXFORM FROM input fields as conditional (the default) and reports on the output from the MODIFY:

```
SET FIXFRMINPUT = COND
-? SET FIXFRMINPUT &FIXF

MODIFY FILE EMPDATA
 FIXFORM FROM TRANS
 MATCH PIN
  ON MATCH UPDATE LN
  ON NOMATCH REJECT
 DATA ON TRANS
END

TABLE FILE EMPDATA
HEADING
" "
"VALUE OF FIXFRMINPUT IS &FIXF "
" "
PRINT PIN LN
 IF PIN FROM '000000010' TO '000000100'
END
```

The output shows that the blank in the transaction file was not used to update the last name in the data source:

```
 VALUE OF FIXFRMINPUT IS COND

PIN        LASTNAME
---        --------
000000010  VALINO
000000020  BELLA
000000030  CASSANOVA
000000040  ADAMS
000000050  ADDAMS
000000060  PATEL
000000070  SANCHEZ
000000080  SO
000000090  PULASKI
000000100  ANDERSON
```

The following procedure sets the FIXFORM FROM input fields as non-conditional and reports on the output from the MODIFY:

```
SET FIXFRMINPUT = NONCOND
-? SET FIXFRMINPUT &FIXF

MODIFY FILE EMPDATA
 FIXFORM FROM TRANS
 MATCH PIN
  ON MATCH UPDATE LN
  ON NOMATCH REJECT
 DATA ON TRANS
END

TABLE FILE EMPDATA
HEADING
" "
"VALUE OF FIXFRMINPUT IS &FIXF "
" "
PRINT PIN LN
 IF PIN FROM '000000010' TO '000000100'
END
```

The output shows that the last name for PIN 000000040 has been updated to contain blanks:

```
 VALUE OF FIXFRMINPUT IS NONCOND

PIN        LASTNAME
---        --------
000000010  VALINO
000000020  BELLA
000000030  CASSANOVA
000000040
000000050  ADDAMS
000000060  PATEL
000000070  SANCHEZ
000000080  SO
000000090  PULASKI
000000100  ANDERSON
```

7 Adapter Enhancements

This chapter describes new features that affect FOCUS adapters (Interfaces).

Topics:

Adabas Dynamic CALLTYPE Setting

Example:

Using Dynamic CALLTYPE

The SET CALLTYPE command enables you to set the data retrieval type per session, file, or segment by switching dynamically between physical and logical reads.

Example: Using Dynamic CALLTYPE

The following examples illustrate the use of variations of dynamic CALLTYPE syntax:

```
ENGINE ADBSINX SET CALLTYPE FIND
```

Uses FIND as the type of data retrieval call for all files.

```
ENGINE ADBSINX SET CALLTYPE RL AFD EMPFILE
```

Retrieves data for the EMPFILE using RL as the type of data retrieval call.

Note that you can issue several commands for different files. Each command must be issued separately. Changes are cumulative. For example, the following commands retrieve data for the EMPFILE using FIND as the type of data retrieval call for a particular ADBS segment (segment S02):

```
ENGINE ADBSINX SET CALLTYPE RL AFD EMPFILE

ENGINE ADBSINX SET CALLTYPE FIND AFD EMP2

ENGINE ADBSINX SET CALLTYPE MIXED AFD VEHICLES

ENGINE ADBSINX SET CALLTYPE FIND AFD EMPFILE SEGNAME S02
```

```
ENGINE ADBSINX SET ?
```

Displays your settings.

```
ENGINE ADBSINX SET CALLTYPE DEFAULT
```

Returns to the default settings. This command resets all SET CALLTYPE commands for all Access Files.

Adabas FETCHJOIN Setting

How to:

Set Adabas FETCHJOIN

You can use the FETCHJOIN command to take advantage of Multifetch efficiencies in Join operations for a session. This feature is particularly useful when you are joining to a base file in which sorting is based on the key field that is used for the join.

Syntax: How to Set Adabas FETCHJOIN

```
ENGINE ADBSINX SET FETCHJOIN {ON|OFF}
```

where:

`ADBSINX`

Indicates the Adapter for Adabas.

`ON`

Sets the FETCHJOIN feature on for the user session. ON is the default value.

`OFF`

Sets the FETCHJOIN feature off for the user session. This value is recommended when joining with a base file that is unsorted by the key field used for joining. This option avoids buffering and reduces processing time.

Adabas SQL NULL Option

Example:

Using the SQL NULL Option

Adabas includes two data definition options, Not Counted (NC) and Not Null or Null Value Not Allowed (NN), for providing SQL-compatible null representation for Software AG's mainframe Adabas SQL Server (ESQ) and other Structured Query Language (SQL) database query languages.

The NC and NN options cannot be applied to fields defined:

- With Adabas null suppression (NU).
- With a fixed-point data type (FI).
- With multiple-values (MU).
- Within a periodic group.
- As group fields.

NC: SQL Null Value Option

Without the Not Counted (NC) option, a null value is either zero or blank depending on the field format.

With the NC option, zeros or blanks specified in the record buffer are interpreted according to the "null indicator" value: either as true zeros or blanks (that is, as "significant" nulls) or as undefined values (that is, as true SQL or "insignificant" nulls).

If the field defined with the NC option has no value specified in the record buffer, the field value is always treated as an SQL null.

Note: On the mainframe platform, subdescriptors and superdescriptors defined with NC=YES and MISSING=ON parameters cannot be used to search for the SQL NULL value, as that causes an Adabas RC 61.

NN: SQL Not Null Option

The Not Null or Null Value Not Allowed (NN) option may only be specified when the NC option is also specified for a data field. The NN option indicates that an NC field must always have a value (including zero or blank) defined; it cannot contain "no value".

Example: Using the SQL NULL Option

If a field with the NC option *does not have* the NN option, then the parameter MISSING=ON is also added to the field definition in the Master File. Subdescriptors and superdescriptors derived from this field will be defined as the fields with the following options: TYPE=NOP, NC=YES, and MISSING=ON. No component fields will be associated with these subdescriptors and superdescriptors.

Sample Master File and Access File contents with the NC option:

Master File:

```
FILENAME=ADANC1, SUFFIX=ADBSINX , $
  SEGMENT=S01, SEGTYPE=S0, $
    FIELDNAME=AA_FIELD, ALIAS=AA, USAGE=I6, ACTUAL=I2, FIELDTYPE=I, $
    FIELDNAME=AB_FIELD, ALIAS=AB, USAGE=I6, ACTUAL=I2, FIELDTYPE=I, MISSING=ON, $
    FIELDNAME=AC_FIELD, ALIAS=AC, USAGE=I6, ACTUAL=I2, FIELDTYPE=I, $
    FIELDNAME=AD_FIELD, ALIAS=AD, USAGE=I6, ACTUAL=I2, MISSING=ON, $
    FIELDNAME=AE_FIELD, ALIAS=AE, USAGE=I6, ACTUAL=I2, $
    FIELDNAME=BA_FIELD, ALIAS=BA, USAGE=A2, ACTUAL=A2, FIELDTYPE=I, $
    FIELDNAME=BB_FIELD, ALIAS=BB, USAGE=A2, ACTUAL=A2, FIELDTYPE=I, MISSING=ON, $
    FIELDNAME=BC_FIELD, ALIAS=BC, USAGE=A2, ACTUAL=A2, FIELDTYPE=I, $
    FIELDNAME=BD_FIELD, ALIAS=BD, USAGE=A2, ACTUAL=A2, MISSING=ON, $
    FIELDNAME=BE_FIELD, ALIAS=BE, USAGE=A2, ACTUAL=A2, $
   GROUP=G1_GROUP, ALIAS=G1, USAGE=A6, ACTUAL=A6, $
    FIELDNAME=CA_FIELD, ALIAS=CA, USAGE=A2, ACTUAL=A2, $
    FIELDNAME=CB_FIELD, ALIAS=CB, USAGE=A2, ACTUAL=A2, MISSING=ON, $
    FIELDNAME=CC_FIELD, ALIAS=CC, USAGE=A2, ACTUAL=A2, $
$ $$$$$$$$$$$$$$$$$$$$$$ Superdescriptor $$$$$$$$$$$$$$$$$$$$$$$$$$ $
    FIELDNAME=S1_FIELD, ALIAS=S1, USAGE=A6, ACTUAL=A6, FIELDTYPE=I, MISSING=ON, $
  $ FIELD=BA_FIELD_S01       ,ALIAS=BA         ,A2    ,A2    ,INDEX=I,$
  $ FIELD=BB_FIELD_S01       ,ALIAS=BB         ,A2    ,A2    ,INDEX=I,$
  $ FIELD=BE_FIELD_S01       ,ALIAS=BE         ,A2    ,A2    ,       ,$
$ $$$$$$$$$$$$$$$$$$$$$$ Superdescriptor $$$$$$$$$$$$$$$$$$$$$$$$$$ $
     FIELDNAME=S2_FIELD, ALIAS=S2, USAGE=A6, ACTUAL=A6, FIELDTYPE=I, MISSING=ON, $
$ FIELD=BA_FIELD_S02       ,ALIAS=BA         ,A2    ,A2    ,INDEX=I,$
$ FIELD=BC_FIELD_S02       ,ALIAS=BC         ,A2    ,A2    ,INDEX=I,$
$ FIELD=BD_FIELD_S02       ,ALIAS=BD         ,A2    ,A2    ,       ,$
$ $$$$$$$$$$$$$$$$$$$$$$$ Subdescriptor $$$$$$$$$$$$$$$$$$$$$$$$$$$ $
      FIELDNAME=S3_FIELD, ALIAS=S3, USAGE=A1, ACTUAL=A1, FIELDTYPE=I, MISSING=ON, $
$ $$$$$$$$$$$$$$$$$$$$$$$ Subdescriptor $$$$$$$$$$$$$$$$$$$$$$$$$$$ $
    FIELDNAME=S4_FIELD, ALIAS=S4, USAGE=A1, ACTUAL=A1, FIELDTYPE=I, $
$
```

Access File:

```
 RELEASE=6, OPEN=YES, $
 SEGNAM=S01, ACCESS=ADBS, FILENO=120, DBNO=3, CALLTYPE=FIND,
   UNQKEYNAME=AA_FIELD, WRITE=YES, $
$   CALLTYPE=RL  ,SEQFIELD=AA_FIELD                        ,$
$ FIELD=AA_FIELD ,TYPE=DSC       ,NC=   ,$
  FIELD=AB_FIELD, TYPE=DSC, NC=YES, $
  FIELD=AC_FIELD, TYPE=DSC, NC=YES, $
  FIELD=AD_FIELD, TYPE=, NC=YES, $
  FIELD=AE_FIELD, TYPE=, NC=YES, $
$ FIELD=BA_FIELD ,TYPE=DSC       ,NC=   ,$
  FIELD=BB_FIELD, TYPE=DSC, NC=YES, $
  FIELD=BC_FIELD, TYPE=DSC, NC=YES, $
  FIELD=BD_FIELD, TYPE=, NC=YES, $
  FIELD=BE_FIELD, TYPE=, NC=YES, $
  FIELD=CB_FIELD, TYPE=, NC=YES, $
  FIELD=CC_FIELD, TYPE=, NC=YES, $
  FIELD=S1_FIELD, TYPE=NOP, NC=YES, $
  FIELD=S2_FIELD, TYPE=NOP, NC=YES, $
  FIELD=S3_FIELD, TYPE=NOP, NC=YES, $
  FIELD=S4_FIELD, TYPE=NOP, NC=YES, $
```

Note: The following processing rules apply for fields defined with the NC=YES and MISSING=ON parameters:

❑ The default value is set to the SQL NULL value when a new record is created:

```
SQL
INSERT INTO ADANC1
    (AA_FIELD,   AB_FIELD,   AC_FIELD,
     BA_FIELD,   BC_FIELD,   BE_FIELD,
     CA_FIELD,   CC_FIELD)
     VALUES (21, 22, 23, 'B1', 'B3', 'B5','A2', 'C2');
END
```

❑ The value could be set to the SQL NULL value when a record is updated:

```
SQL
  UPDATE ADANC1 SET AB_FIELD=NULL
  WHERE  AA_FIELD = 21;
END
```

❑ These fields can be used in search criteria for selecting records that either *have or do not have* SQL NULL values, as in the following examples:

```
SQL
  SELECT AA_FIELD, AB_FIELD, AC_FIELD, AD_FIELD, AE_FIELD FROM ADANC1
  WHERE  AB_FIELD IS NULL AND AA_FIELD GE 10;
END

SQL
  SELECT AA_FIELD, AB_FIELD, AC_FIELD, AD_FIELD, AE_FIELD FROM ADANC1
  WHERE  AB_FIELD IS NOT NULL;
END

TABLE FILE ADANC1R
  PRINT AA_FIELD AB_FIELD AC_FIELD AD_FIELD AE_FIELD G1_GROUP
  WHERE  AB_FIELD EQ MISSING;
END
```

❑ Subdescriptors and superdescriptors can be used only in search criteria. Their values cannot be reported. Here are several examples:

```
SQL
  SELECT AA_FIELD, AB_FIELD, AC_FIELD, AD_FIELD, AE_FIELD FROM ADANC1
  WHERE  S1_FIELD = 'A1A2A4' AND AA_FIELD > 10;
END

SQL
  SELECT AA_FIELD, AB_FIELD, AC_FIELD, AD_FIELD, AE_FIELD FROM ADANC1
  WHERE  S1_FIELD IS NOT NULL AND AA_FIELD > 10;
END
```

Note: On the mainframe platform, subdescriptors and superdescriptors defined with the NC=YES and MISSING=ON parameters cannot be used to search for an SQL NULL value, as that causes an Adabas RC 61.

PASSRECS Setting for Adapters for Adabas, IMS, and VSAM

PASSRECS returns the number of rows affected by a successfully executed SQL Passthru INSERT, UPDATE, or DELETE command.

Syntax: How to Obtain the Number of Rows Inserted, Updated, or Deleted

```
ENGINE INT SET PASSRECS {ON|OFF}
```

where:

`INT`

Indicates that the PASSRECS setting in this command will be applied globally to all adapters that support SQL INSERT, UPDATE, and DELETE commands.

`ON`

Provides the number of rows affected in the application program SCB count member after the successful execution of an SQL Passthru INSERT, UPDATE, or DELETE command. ON is the default value.

`OFF`

Provides no information after the successful execution of an SQL Passthru INSERT, UPDATE, or DELETE command.

In addition, the adapter updates the &RECORDS system variable with the number of rows affected. You can access this variable using Dialogue Manager.

VSAM RRDS Support

How to:

Enable VSAM RRDS Support

Example:

Applying RRN Key Values When Working With Records in VSAM RRDS Files

The Adapter for VSAM supports both direct and sequential retrieval of fixed- and variable-length records from RRDS files, as well as modification of those records. RRDS files are VSAM files that are accessed through Relative Record Number keys (RRNs) and are processed in basically the same manner as key-sequenced (KSDS or KSEQ) VSAM files.

The RRN field that serves as the primary (unique) key for the segment follows all real fields in the physical root segment of the Master File and establishes the sequence for the records. The RRN field description must contain the following attributes:

```
ALIAS=RRN, USAGE=I10, ACTUAL=I4
```

Records in RRDS files are processed sequentially if the request does not provide a screening condition or join for the RRN-associated field.

Syntax: How to Enable VSAM RRDS Support

```
FILE=RRDSVSAM, SUFFIX=VSAM, DATASET=PGMYAN.RRDSVSAM.CLUSTER, $
SEGNAME=ROOT ,SEGTYPE=S0 ,$
FIELD=CODE , ,USAGE=A2 ,ACTUAL=A2 ,$
FIELD=NAME , ,USAGE=A10 ,ACTUAL=A10 ,$
FIELD=VALUE , ,USAGE=P2 ,ACTUAL=Z1 ,$
FIELD=ACCESS_KEY ,ALIAS=RRN ,USAGE=I10 ,ACTUAL=I4 ,$
```

Example: Applying RRN Key Values When Working With Records in VSAM RRDS Files

The following examples show how to view and apply unique Relative Record Number (RRN) key values when working with records in RRDS files.

Printing the records with their key values

```
TABLE FILE RRDSVSAM
PRINT CODE NAME VALUE ACCESS_KEY
END
```

Screening records by unique key values

```
TABLE FILE RRDSVSAM
PRINT CODE NAME VALUE ACCESS_KEY
IF ACCESS_KEY  GE 5
IF ACCESS_KEY  LE 15
END
```

Retrieving records by unique key values

```
TABLE FILE RRDSVSAM
PRINT CODE NAME VALUE ACCESS_KEY
IF ACCESS_KEY  EQ 5 OR 7 OR  9
END
```

Modifying records by unique key values

```
MODIFY FILE RRDSVSAM ECHO
FREEFORM ACCESS_KEY
MATCH ACCESS_KEY
ON NOMATCH COMPUTE CODE = 'A1';
ON NOMATCH COMPUTE NAME = 'RECORD 20';
ON NOMATCH INCLUDE
ON MATCH REJECT
DATA
ACCESS_KEY=20
END
```

Updating matching records in RRDS files by unique key values

```
MODIFY FILE RRDSVSAM ECHO
FREEFORM ACCESS_KEY
MATCH ACCESS_KEY
ON NOMATCH REJECT
ON MATCH
TYPE "CODE: <D.CODE>, NAME: <D.NAME>, VALUE: <D.VALUE>";
ON MATCH COMPUTE VALUE = 9;
ON MATCH UPDATE VALUE
DATA
ACCESS_KEY=20
END
```

Deleting RRDS records based on unique key values

```
MODIFY FILE RRDSVSAM ECHO
FREEFORM ACCESS_KEY
MATCH ACCESS_KEY
ON NOMATCH REJECT
ON MATCH
TYPE "CODE: <D.CODE>, NAME: <D.NAME>, VALUE: <D.VALUE>";
ON MATCH DELETE
DATA
ACCESS_KEY=7
END
```

HOLD FORMAT SAME_DB

In this section:

Column Names in the HOLD File

Primary Keys and Indexes in the HOLD File

How to:

Save Report Output as a Temporary Table

Reference:

Temporary Table Properties for SAME_DB Persistence Values

You can create a report output file—that is, a HOLD file—as a native DBMS temporary table. This increases performance by keeping the entire reporting operation on the DBMS server, instead of downloading data to your computer and then back to the DBMS server.

For example, if you temporarily store report output for immediate use by another procedure, storing it as a temporary table instead of creating a standard HOLD file avoids the overhead of transmitting the interim data to your computer.

The temporary table columns are created from the following report elements

- ❏ Display columns
- ❏ Sort (BY) columns
- ❏ COMPUTE columns

except for those for which NOPRINT is specified.

The temporary table that you create from your report will be the same data source type (that is, the same DBMS) as the data source from which you reported. If the data source from which you reported contains multiple tables, all must be of the same data source type and reside on the same instance of the DBMS server.

You can choose between several types of table persistence.

You can create extract files as native DBMS tables with the following adapters:

- ❏ DB2
- ❏ Oracle
- ❏ Teradata

Syntax: How to Save Report Output as a Temporary Table

The syntax to save report output as a native DBMS temporary table is

```
ON TABLE HOLD [AS filename]FORMAT SAME_DB [PERSISTENCE persistValue]
```

where:

filename

Specifies the name of the HOLD file. If you omit AS *filename*, the name of the temporary table defaults to “HOLD”.

Because each subsequent HOLD command overwrites the previous HOLD file, it is recommended to specify a name in each request to direct the extracted data to a separate file, thereby preventing an earlier file from being overwritten by a later one.

PERSISTENCE

Specifies the type of table persistence and related table properties. This is optional for DBMSs that support volatile tables, and required otherwise. For information about support for volatile tables for a particular DBMS, see *Temporary Table Properties for SAME_DB Persistence Values*, and consult your DBMS vendor’s documentation.

`persistValue`

Is one of the following:

`VOLATILE`

Specifies that the table is local to the DBMS session. A temporary synonym—a Master File and Access File—is generated automatically; it expires when the server session ends.

This is the default persistence setting for all DBMSs that support volatile tables.

For information about support for the volatile setting, and about persistence and other table properties, for a particular DBMS, see *Temporary Table Properties for SAME_DB Persistence Values*, and consult your DBMS vendor's documentation.

`GLOBAL_TEMPORARY`

Specifies that while the table exists, its definition will be visible to other database sessions and users though its data will not be. A permanent synonym—a Master File and Access File—is generated automatically.

For information about support for the global temporary setting, and about persistence and other table properties, for a particular DBMS, see *Temporary Table Properties for SAME_DB Persistence Values*, and consult your DBMS vendor's documentation.

`PERMANENT`

Specifies that a regular permanent table will be created. A permanent synonym—a Master File and Access File—is generated automatically.

Reference: Temporary Table Properties for SAME_DB Persistence Values

The following chart provides additional detail about persistence and other properties of temporary tables of different data source types that are supported for use with HOLD format SAME_DB.

DBMS	VOLATILE	GLOBAL_TEMPORARY
DB2	DB2: a volatile table is created using the DECLARE GLOBAL TEMPORARY TABLE command with the ON COMMIT PRESERVE ROWS option. Declared global temporary tables persist and are visible only within the current session (connection). SESSION is the schema name for all declared global temporary tables.	DB2 Release 7.1 and up for z/OS only: a global temporary table is created using the CREATE GLOBAL TEMPORARY TABLE command. The definition of a created global temporary table is visible to other sessions, but the data is not. The data is deleted at the end of each transaction (COMMIT or ROLLBACK command). The table definition persists after the session ends.
Oracle	This type of table is not supported by Oracle.	The table's definition is visible to all sessions; its data is visible only to the session that inserts data into it. The table's definition persists for the same period as the definition of a regular table.
Teradata	A volatile table definition and data are visible only within the session that created the table and inserted the data. The volatile table is created with the ON COMMIT PRESERVE ROWS option.	A global temporary table persists for the same duration as a permanent table. The definition is visible to all sessions, but the data is visible only to the session that inserted the data. The global temporary table is created with the ON COMMIT PRESERVE ROWS option.

Column Names in the HOLD File

Each HOLD file column is assigned its name:

1. From the AS name specified for the column in the report request.
2. If there is no AS name specified, the name is assigned from the alias specified in the synonym. (The alias is identical to the column name in the original relational table.)

3. In all other cases, the name is assigned from the field name as it is specified in the synonym.

Primary Keys and Indexes in the HOLD File

A primary key or an index is created for the HOLD table. The key or index definition is generated from the sort (BY) keys of the TABLE command, except for undisplayed sort keys (that is, sort keys for which NOPRINT is specified). To determine whether a primary key or an index will be created:

1. If these sort keys provide uniqueness and do not allow nulls (that is, if in the synonym the column's MISSING attribute is unselected or OFF), and if the DBMS supports primary keys on the type of table being created, a primary key is created.
2. If these sort keys provide uniqueness but either

    a. some of the columns allow nulls

    b. the DBMS does not support primary keys on the type of table being created

    then a unique index is created.
3. If these sort keys do not provide uniqueness, a non-unique index is created.
4. If there are no displayed sort keys (that is, no sort keys for which NOPRINT has not been specified), no primary key or index is created.

Adapter for Oracle: Increased Length of Column Descriptions

The adapter supports column descriptions of up to 255 characters.

Oracle 10g Support

The Adapter for Oracle now supports Oracle Release 10g. To install this version of the adapter, you must edit and submit member G10FORA in the FOCSQL.DATA data set. This member links Oracle with FOCUS.

Oracle 10g is stored in a PDSE data set. To link Oracle 10g to FOCUS, you must link it to a version of the FOCUS FOCLIB.LOAD data set that is also stored in a PDSE data set.

To run an interactive FOCUS session that accesses Oracle 10g, you must have a region size that is at least 32 megabytes.

Following is the G10FORA member of the FOCSQL.DATA data set. Add a job card, make the necessary edits, and submit this JCL to install the adapter:

```
//job card goes here
//*********************************************************************
//* Name:     G10FORA JCL
//*
//* Function: Linkedit the MVS FOCUS/ORACLE 10  interface
//*
//* Substitutions:-Change "qualif" into the high level qualifier
//*                for your FOCUS datasets.
//*               -Change "qualpdse.LOAD" to the name of your
//*                FOCUS PDSE load library.
//*               -Change "oraqual.SQLLIB" TO THE NAME OF YOUR
//*                Oracle SQL library.
//*               -Change "libprfx" to the high level qualifier
//*                of your linker library (like CEE)
//*               -Change "plang" to the name of your linker
//*                message name (like EDCPMSGE)
//*********************************************************************
//*
//LKED1   EXEC PGM=IEWL,PARM='LIST,NOXREF,LET,RENT'
//OBJECT   DD DISP=SHR,DSN=qualif.FOCSQL.DATA
//SYSMODIN DD DISP=SHR,DSN=qualif.FOCSQL.LOAD
//SYSLMOD  DD DISP=SHR,DSN=qualpdse.LOAD
//SYSUT1   DD UNIT=SYSDA,SPACE=(100,(50,50))
//SYSPRINT DD SYSOUT=A
//LKED1.SYSLIN   DD *
  INCLUDE SYSMODIN(OR8FOC)
  INCLUDE OBJECT(OR8FOC)
  NAME ORAFOC(R)
/*

//*
//LKED2   EXEC PGM=IEWL,
//   PARM='LIST,NOXREF,LET,RENT,AMODE=31,RMODE=ANY,UPCASE'
//SYSLIB   DD DISP=SHR,DSN=libprfx.SCEELKED
//SYSMSGS  DD DSN=libprfx.SCEEMSGP(plang),DISP=SHR
//OBJECT   DD DISP=SHR,DSN=qualif.FOCSQL.DATA
//ORAOBJ   DD DISP=SHR,DSN=oraqual.SQLLIB
//SYSLMOD  DD DISP=SHR,DSN=qualpdse.LOAD
//SYSUT1   DD UNIT=SYSDA,SPACE=(100,(50,50))
//SYSPRINT DD SYSOUT=A
//LKED2.SYSLIN   DD *
  INCLUDE OBJECT(OR8PS2)
  INCLUDE OBJECT(ORAPAS10)
  INCLUDE ORAOBJ(ORASTBL)
  NAME ORAPAS2(R)
/*
```

where:

qualif

Is the high-level qualifier for your production FOCUS data sets.

qualpdse

Is the high-level qualifier for your FOCUS load library in PDSE format.

oraqual

Is the high-level qualifier for your Oracle SQL library.

libprfx

Is the high-level qualifier for your link edit load library.

plang

Is the linkage editor message member name.

Relational Adapters: Controlling Optimization of Calculations

Calculations can be processed differently in different RDBMSs and operating environments. If you want FOCUS to handle calculations instead of the RDBMS, you can issue the The SQL SET OPTIMIZATION NOAGGR command. This command disables optimization of calculations (DEFINE fields) without disabling optimization of join and sort operations.

Syntax: How to Disable Optimization of Calculations

```
SQL [engine] SET OPTIMIZATION NOAGGR
```

where:

engine

Indicates the target RDBMS. Valid values are DB2, SQLORA, SQLDS, SQLIDMS, or SQLDBC. You can omit this value if you previously issued the SET SQLENGINE command.

Calling a Teradata Macro or Stored Procedure Using SQL Passthru

Example:

Calling a Macro

Calling a Stored Procedure

Sample Teradata Stored Procedure

SQL Passthru is supported for Teradata macros and stored procedures.

Macros need to be developed within Teradata using the CREATE or REPLACE MACRO command. Procedures need to be developed within Teradata using the CREATE PROCEDURE command.

You must call a macro in the same transaction mode in which it was compiled. To find out which transaction mode is currently in effect, issue the HELP SESSION command:

```
SQL SQLDBC HELP SESSION;
TABLE FILE SQLOUT PRINT TRANSACTION_SEMANTICS
END
```

Before you can call a stored procedure or a macro, you must set the connection accordingly, by issuing the SET MACRO command

```
SQL SQLDBC SET MACRO {ON|OFF}
```

where:

ON

Enables one to call a macro. This is the default.

OFF

Enables one to call a stored procedure.

Example: Calling a Macro

This is an example of the syntax for calling a macro:

```
ENGINE SQLDBC
EX SAMPLE PARM1,PARM2,PARM3...;
TABLE FILE SQLOUT
END
```

Example: Calling a Stored Procedure

The supported syntax to call a stored procedure is shown below.

```
ENGINE SQLDBC
EX SAMPLE PARM1,PARM2,PARM3...;
TABLE FILE SQLOUT
END
```

When using the adapter with:

- ❑ **ODBC,** all scalar parameters (IN, OUT, and INOUT) are supported.
- ❑ **CLI,** scalar IN parameters, and INOUT parameters in IN mode, are supported.

Example: Sample Teradata Stored Procedure

```
CREATE OR REPLACE PACKAGE pack1 AS
TYPE nfrectype IS RECORD (
employee NF29005.EMPLOYEE_ID5%TYPE,
ssn5     NF29005.SSN5%TYPE,
l_name   NF29005.LAST_NAME5%TYPE,
f_name   NF29005.FIRST_NAME5%TYPE,
birthday NF29005.BIRTHDATE5%TYPE,
salary   NF29005.SALARY5%TYPE,
joblevel NF29005.JOB_LEVEL5%TYPE);
TYPE nfcurtype IS REF CURSOR RETURN nfrectype ;
PROCEDURE proc1(c_saltable IN OUT nfcurtype);
END pack1 ;
/
CREATE OR REPLACE PACKAGE BODY pack1 AS
PROCEDURE proc1 (c_saltable IN OUT nfcurtype)
IS
BEGIN
OPEN c_saltable FOR SELECT
EMPLOYEE_ID5,SSN5,LAST_NAME5,FIRST_NAME5,BIRTHDAT
E5,SALARY5,JOB_LEVEL5 FROM NF29005;
END proc1 ;  -- end of procedure
END pack1; -- end of package body
/
```

8 Raised Limits

This chapter describes limits that have been increased in this release of FOCUS.

For information on the increased number of partitions for a FOCUS or XFOCUS database, see Chapter 5, *Database Enhancements*.

Topics:

Increased Number of Segments in a Structure

Non-FOCUS data sources, JOIN structures used for reporting, and COMBINE structures used for MODIFY can have a maximum of 256 segments. XFOCUS data sources can have 256 segments, but when they are included in COMBINE structures used for MODIFY requests, the limit is 192 (segments plus indices).

The total number of segments in the structure includes segments defined as static cross references (SEGTYPE = KU or KM) in the Master File. Dynamic cross references (SEGTYPE = DKM, DKU, KL, or KLU) are not included in the count.

All other limits such as the number of fields in a Master File, the total length of all fields, and the number of JOINs or COMBINEs allowed are still in effect.

The number of segments in a single FOCUS data source is 64. However, FOCUS data sources can participate in JOIN or COMBINE structures with up to 256 segments. The maximum number of segments plus indices for a single FOCUS data source is 191.

Using an indexed view reduces the maximum number of segments plus indices to 191 for the structure being used. If AUTOINDEX is ON, you may be using an indexed view without specifically asking for one.

Reference: Usage Notes for 256 Segments

Maintain supports a maximum of 64 segments for all file types (FOCUS and non-FOCUS).

Increased Length for HOLD Files in FOCUS Format

Reference:

Usage Notes for HOLD FORMAT FOCUS With Unique Child Segments

Example:

Creating a FORMAT FOCUS Data Source With Unique Child Segments

A FOCUS segment cannot be longer than a FOCUS page, 3968 (4096 minus overhead) bytes. An XFOCUS segment can be up to 16K bytes. In either case, a data source path cannot be longer than 32K bytes.

In prior releases, any HOLD FORMAT FOCUS or XFOCUS request that specified more than a page of data generated the following message, and the data source was not created:

```
(FOC397)     LENGTH OF FIELDS ON FOCUS DATA PATH EXCEEDS 32K BYTES:
```

The sum of the field lengths on a data source path exceeds the maximum, reduce the size or number of fields.

Now, when you create a FOCUS or XFOCUS data source using the HOLD command, data that is too long to become a single segment will become a parent segment with unique child segments. For a FOCUS data source, the fields will be grouped into normal FOCUS page size segments and added as unique segments up to the total maximum of 32K of data. For an XFOCUS data source, the root segment can hold the first 16K of data, and additional data up to the 32K total, will be placed in a single unique segment.

The FOC397 message will no longer be generated as long as the total length of the data for a segment path is less than 32K.

Each BY field in the request becomes a key in the root segment. Therefore, no BY field can be in the portion of the data placed in a unique child. If a BY field occurs in the portion of the data past the length of the root segment, the following message is generated:

```
(FOC400)     KEY FIELDS LENGTH IN SEGMENT EXCEEDED MAXIMUM:
```

The key fields in the segment exceeded maximum

Reference: Usage Notes for HOLD FORMAT FOCUS With Unique Child Segments

❏ Addition of unique segments impacts both performance and volume.

❏ BY fields must all occur in the portion of the data assigned to the root segment.

For more information about HOLD FORMAT FOCUS, see the *Creating Reports* manual.

Example: Creating a FORMAT FOCUS Data Source With Unique Child Segments

The following request creates a version of the MOVIES data source with data longer than 3968 bytes:

```
SET ASNAMES=ON
DEFINE FILE MOVIES
NEWTITLE/A1024=TITLE;
END
TABLE FILE MOVIES
PRINT NEWTITLE AS TITLE1
      NEWTITLE AS TITLE2
      NEWTITLE AS TITLE3
      NEWTITLE AS TITLE4
      NEWTITLE AS TITLE5
ON TABLE HOLD AS GIANTMOV FORMAT FOCUS
END
```

The Master File generated by this request has one unique child segment:

```
FILENAME=GIANTMOV, SUFFIX=FOC     , $
  SEGMENT=SEG01, SEGTYPE=S1, $
    FIELDNAME=FOCLIST, ALIAS=E01, USAGE=I5, $
         FIELDNAME=TITLE1, ALIAS=E02, USAGE=A1024,  TITLE='TITLE1', $
         FIELDNAME=TITLE2, ALIAS=E03, USAGE=A1024,  TITLE='TITLE2', $
         FIELDNAME=TITLE3, ALIAS=E04, USAGE=A1024,  TITLE='TITLE3', $
  SEGMENT=SEG02, SEGTYPE=U, PARENT=SEG01, $
       FIELDNAME=TITLE4, ALIAS=E05, USAGE=A1024,  TITLE='TITLE4', $
       FIELDNAME=TITLE5, ALIAS=E06, USAGE=A1024,  TITLE='TITLE5', $
```

Increased Name Length for Dialogue Manager Variables

Dialogue Manager variable names on z/OS and z/VM can be up to 100 characters.

Increased Number of Amper Variables

An unlimited number of Dialogue Manager variables can be used in a procedure.

Raised Limit for Number of FML Rows

Reference:

Error Messages

The limit on FML tag values is now based on available memory rather than on a hardcoded value to service larger FML requests. There is no new syntax and users interact with FML by entering FML requests using an editor as before.

Reference: Error Messages

If it is determined there is no more available memory, an internal routine displays one of the following messages before exiting the product:

```
Reqfor: out of memory
```

This indicates either too many labels or too much memory required.

```
Rectok: out of memory
```

Generated request cannot be parsed because not enough memory.
(This is not an FML-specific message.)

Increased Number of Joins

The number of joins that can be in effect at one time for one data source has been raised to 63. The joined structure can have a total of up to 64 segments. The total length of all fields can be up to 32K and the number of fields in a data source can be up to 3072.

For example, you can have 63 joins between 64 one-segment data sources or three joins between four 16-segment data sources, or any combination that does not exceed 64 segments.

If you are using a relational data adapter, the Relational Database Management System (RDBMS) may have its own limit for the number of joins allowed in an SQL statement. If you create a joined structure that exceeds the number of JOINs allowed by the RDBMS but is within the number of JOINs allowed by FOCUS, the join is processed by FOCUS, but not optimized (passed to the RDBMS for processing).

Increased Number of Files in a MATCH FILE Request

You could previously merge up to six different data sources or merge data from the same data source up to six times per request. That limit has been raised to 16, allowing you to merge up to 16 data sources in a single MATCH request.

The manner in which MATCH merges the data depends on the request context, or more specifically, the order in which the data sources, BY fields, and display commands are mentioned, as well as on the number of verb objects and BY phrases employed. As before, the following general MATCH conditions apply:

- ❏ You can use a total of 32 BY phrases and the maximum normal number of display fields in each MATCH request. You must specify at least one BY field for each file named in the MATCH request.
- ❏ DEFINE commands can be used.
- ❏ Up to 32 sort sets are supported, including the normal number of common sort fields.
- ❏ If used with MATCH, SET HOLDLIST behaves as if HOLDLIST were set to ALL.
- ❏ The ACROSS and WHERE TOTAL phrases are not permitted and you cannot use COMPUTE or BY HIGHEST in a MATCH request.

There is no new syntax for this feature.

Reference: Usage Notes for MATCH FILE Limit

The limit of 16 files represents the number of files supported in the most complex MATCH FILE requests. Simpler requests may be able to merge a higher number of files.

Increased Length for DESCRIPTION Attribute in a Master File

The length of the DESCRIPTION attribute for a field definition in a Master File has been raised to 2K.

Increase to the Total Length of All fields

The length of the LINREC and DATREC buffers has increased to 64K. Therefore, the total length of all fields read in a request can be up to 64K bytes. This includes fields in the Master File that are used in the request, fields created with the DEFINE and COMPUTE commands, and fields created internally as a result of internal computations or field reformatting. The total length of fields in the report output has not increased.

The number of fields allowed has not changed. All existing compiled MODIFY procedures should work without recompiling them.

Increased Length of DBA Passwords

DBA passwords can be up to 64 characters in length.

SET MAXLRECL = 65536

How to:

Enable Reading a File with a 65536 Byte Record Length

MAXLRECL indicates the largest actual file record length that FOCUS can read. The limit for MAXLRECL is 65536 bytes, allowing the user to read a record twice as large as the length of the internal matrix, which is limited to 32K. Therefore any request has this processing and output limitation.

Syntax: How to Enable Reading a File with a 65536 Byte Record Length

```
SET MAXLRECL = 65536
```

9 Logging FOCUS Usage: FOCLOG

Starting in FOCUS 7.6.3, FOCLOG is a tool for recording and analyzing the use of FOCUS for your entire site. It comes packaged with a set of standard analytical reports that allow you to interrogate FOCUS usage—identify usage spikes and redundancies, detect large report requests, analyze time-of-day usage trends, and monitor ad hoc versus scheduled requests for each user. It even allows you to analyze the environmental conditions of the query such as use of joins, cross references, combines, MSO or SU. In addition, it collects and reports on statistics such as the number of data rows extracted and number of lines on the report output.

FOCLOG is invisible and non-intrusive to your production applications, whether batch or online. Rolled up and sorted in creative ways, the captured data provides the insight a site coordinator or manager needs to gain a clear picture of FOCUS usage and to target areas that could require adjustment, consolidation, or expansion.

Topics:

Overview of FOCLOG

In this section:

FOCLOG is a facility for logging FOCUS usage that was designed to have a negligible impact on FOCUS applications and to make it easy to analyze the collected data.

The cost of logging has been made so low as to be insignificant, so FOCUS usage can now be monitored continually, not just for selected periods. This is because the log is a simple sequential file to which usage data gathered during the FOCUS session or batch job is only appended when the FOCUS session ends.

Updates to the log from concurrently executing FOCUS sessions or batch jobs are serialized through a standard systems-wide z/OS Enqueue macro. The systems-wide scope allows a single log to gather usage data from users in different LPARS on the same machine.

The log is a physical sequential file with multiple record types that is easily read by FOCUS using a distributed Master File. Multiple logs can be logically concatenated so they are viewed as a single entity, and analyzed right where they are. Alternatively, since all the data is in character format, it can easily be moved to another platform—for example a laptop—where a more graphical analysis can be done using WebFOCUS.

How Logging is Implemented

The FOCLOG facility is incorporated into FOCUS. However, logging is only triggered if at the start of execution, FOCUS detects the presence of member FOCUSLOG in any of the data sets allocated to DDNAME ERRORS on z/OS or the FOCUSLOG ERRORS file on z/VM. This file in turn names the log data set itself. The ERRORS DDNAME is already allocated through JCL in all production FOCUS environments—FOCUS will not execute without it.

To inhibit logging, you can delete or rename the FOCUSLOG file or delete or rename the reference to the log file within the FOCUSLOG file.

When logging is triggered, usage data is collected and kept in memory as the application proceeds. The log itself is not opened until the FOCUS session or batch job ends. If FOCUS abends in the course of execution, or if it is stopped by the operator, the usage data will not be logged for those user sessions active at the time of the stoppage.

Extreme precautions have been taken to ensure that logging difficulties will not affect the application. If the log cannot be written for whatever reason (for example, it is full, the user has no write privilege to it, there has been an I/O error), the only effect on the application will be that the usage data will not be logged. This is true even if the logging difficulty results in an abend: FOCUS will recover and terminate normally.

Failures to log are not signaled through FOCUS-generated error messages.

On z/OS, if the logging failure caused an abend from which FOCUS recovered—for example a B37 abend caused by a log full condition—the occurrence of the abend will be recorded in the batch job step statistics or in your TSO log, but logging failures which do not cause an abend will not be recorded in any way.

On z/VM, your SFS (Shared File System) Administrator can see the log full condition recorded in the Filepool Console log.

The z/OS ISPF or z/VM FILELIST utility can be used to examine the date and time of the last update of the log.

The Log Data Set

On z/OS, the first 44 bytes of the first record of the FOCUSLOG member in the concatenation of data sets allocated to DDNAME ERRORS contains the fully qualified data set name of the current log file.

On z/VM, the first record of the FOCUSLOG ERRORS file contains the fully qualified name of the SFS directory.

Other records in the FOCUSLOG file are ignored, so they can safely be used to retain the names of prior logs. The default is to create short logs; long logs are created by following the log file name with the keyword DETAIL. This keyword gives you everything recorded in the short log and adds the data set or file names for the FOCEXEC, Master File, and Access File used in each request.

The site administrator must ensure that the LOG file is write-accessible to all FOCUS users and protected against archiving—FOCUS will not wait until an archived log is restored, rather it will skip logging instead. Therefore, the log file should *not* be managed by DFSMS.

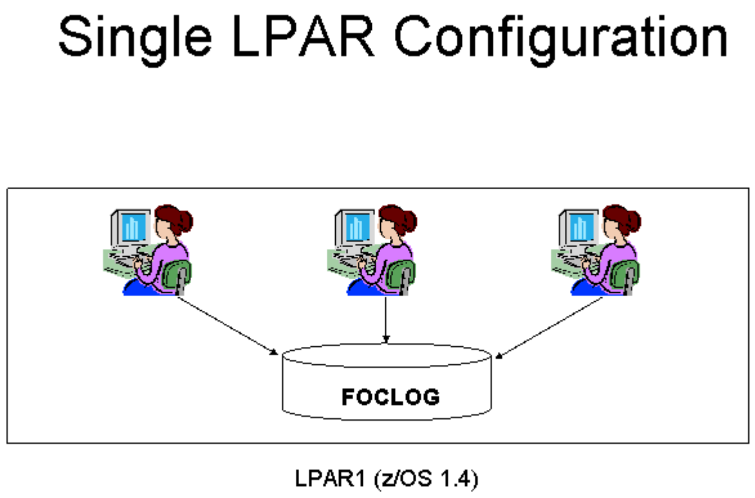
The log consists of four types of records:

Type of record	Size in bytes	Number of Occurrences
Session	168	One per FOCUS session or batch job
Command	129 or 173	One per FOCUS command that accesses data
File	28 or 116	One per MASTER file referenced by the command
Dataset	68	One per data set still allocated at the end of the job

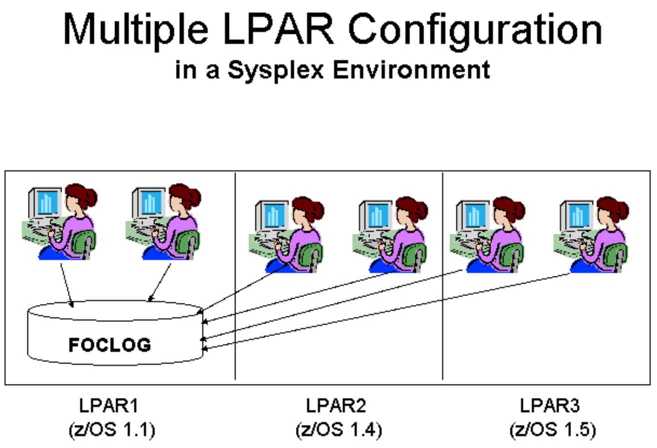
The detailed description of each record is in the distributed FOCLOG Master File and in *Information Captured in the FOCLOG File* on page 369. Long and short logs have identical Master Files; in the short log the Master File, Access File, and FOCEXEC data set names will appear blank.

Sample FOCLOG Configuration Scenarios

In the following diagram of a single LPAR configuration, one FOCLOG file records the usage of all FOCUS users in that LPAR:



In a site configuration where FOCUS resides in several LPARS, even with different OS levels, a single FOCLOG file can reside in any one of the LPARs.



Implementing FOCLOG

In this section:

The person implementing FOCLOG should be a system administrator with the ability to create and allocate data sets and who can assign RACF rules. Knowledge of FOCUS is not required.

The modules, FOCEXECs, and Master File needed to run FOCLOG and the FOCLOG reports are distributed as part of the standard FOCUS libraries. No installation steps are required. The administrator only needs to create a log file and activate it by specifying the fully qualified name of the SFS directory in the FOCUSLOG file.

If your site standards require you to implement new features in a test environment, create a temporary FOCUSLOG ERRORS file.

- ❑ On z/OS, create a copy of your FOCUS production CLIST and concatenate your ERRORS library in front of the FOCUS production ERRORS library.
- ❑ On z/VM, place your FOCUSLOG ERRORS file on a private minidisk that is higher in the search order than the FOCUS production minidisk.

Create a temporary log file and place its fully qualified SFS directory name in your private version of the FOCUSLOG ERRORS file.

After validating the FOCLOG implementation, remove your private FOCUSLOG file from your CLIST or minidisk.

Overview of FOCLOG Implementation

Reference:

Usage Notes for FOCLOG

These steps provide a high level overview of the FOCLOG implementation process:

1. Allocate the FOCLOG data set.
2. Activate FOCLOG.
3. Validate the FOCLOG configuration.
4. Move FOCLOG to your production FOCUS environment.
5. Run the reports supplied with FOCLOG after collecting log data for whatever period you deem appropriate.

Reference: Usage Notes for FOCLOG

- To avoid archiving the FOCLOG data set, it should not be managed by DFSMS.
- To keep the FOCLOG file a manageable size, the log was designed to capture only those commands that allow you to analyze the function of FOCUS applications at your site. Therefore, by design, only the following commands are captured in the log file: RUN (to run a compiled MODIFY), MAINTAIN, CREATE, FSCAN (as FOC$SCAN), GRAPH, HOLD, MATCH FILE, MODIFY, RETYPE, SCAN, TABLE, and TABLEF. Other commands issued during one of these requests (for example, JOIN or CHECK FILE) are not captured in the log file. However, if a JOIN or cross reference is in effect, all of the joined files are listed in the log after a TABLE request that references the join. MATCH FILE captures only the first file.
- The following commands are not captured in the log file: REBUILD, JOIN, CHECK FILE, MORE.
- In the event that the log file becomes full, recording stops for all users. To replace the log file, you can rename the full log file and allocate another empty file with the name of the original log file. Logging will resume the next time a user session ends (at FIN).

Allocating the FOCLOG Log File

How to:

Allocate the FOCLOG Log File on z/OS

Estimate the Size of the Log File

You must have a log file allocated and reference it in the FOCUSLOG ERRORS file in order to initiate logging. You should first determine the size of the log file required as described in *How to Estimate the Size of the Log File* on page 357.

Procedure: How to Allocate the FOCLOG Log File on z/OS

1. Allocate a physical sequential file with LRECL 256 and RECFM VB using the size estimate you determined in Step 1.

   Edit and submit the following job to allocate the FOCLOG log file after adding an appropriate job card:

   ```
   //* add job card here
   //FLALLOC   EXEC PGM=IEFBR14
   //FOCLOGF   DD DSN=flhlq.FOCLOG.DATA,DISP=(NEW,CATLG),
   //              VOL=SER=volser,UNIT=unit,SPACE=(CYL,(p,s)),
   //              DCB=(RECFM=VB,LRECL=256,BLKSIZE=12288),DSORG=PS
   /*
   ```

   where:

   flhlq

   Is the high level qualifier for the FOCLOG file.

   unit

   Is a valid DASD unit for the FOCLOG file.

   volser

   Is the volume serial number of the unit on which you want to store the FOCLOG file.

   p

   Is the primary space allocation in cylinders.

   s

   Is the secondary space allocation in cylinders.

2. Using your security interface (for example, RACF), authorize all of your FOCUS users in the entire sysplex to have write access across the sysplex to this file.

The following are suggestions for naming conventions for your log file:

- ❑ Create a separate log for each day of the week, for example FOCLOGMO, FOCLOGTU, FOCLOGWE, FOCLOGTH, FOCLOGFR, FOCLOGSA, and FOCLOGSU.
- ❑ Use a numbering convention such as FOCLOG1, FOCLOG2, FOCLOG3, and so on, to name multiple log files.
- ❑ Name log files by system and date, such as S1091507. Where S1 represents System number 1 and 091507 represents the day of the year (in this example, September 15, 2007).
- ❑ Allocate the log file as a Generation Data Group (GDG) in which case the system creates a naming convention by using index numbers for each generation.

Procedure: How to Estimate the Size of the Log File

The size of the log file required depends on factors such as the number of users and FOCUS applications in your environment, plus the length of time for which you wish to capture information.

Consider these suggestions when estimating its size:

Estimating user session statistics. A sample user session lasting one hour and containing 26 DD allocations running 25 TABLE requests will produce approximately 77 output records to the log. There will be one session record for the user session, 26 DSNS records (one per DD allocation), and 50 records from the TABLE requests (2 per request). Assuming that the average number of bytes per record is 50, this sample user session will produce 3850 bytes of output in the log.

One cylinder on an IBM 3390-3 device contains 737280 bytes. Therefore, approximately 190 user sessions of this size would fill one cylinder (737280 / 3850).

To apply this calculation at your site, you must evaluate the number of allocations in your production FOCUS CLIST/JCL and estimate how many TABLE requests each session is likely to run in an hour. Then use the following formulas to estimate the size of the log file:

```
bytes_per_hr = no_of_users * ((1+ number_of_allocs + (2 *
avg_no_of_TABLEs)) * 50)

user_sessions_per_cylinder =  737280  / bytes_per_hr
```

Determine how many user sessions you wish to log per day, week, or month and allocate cylinders accordingly. For example, using the sample session described above, eight cylinders would be adequate for collecting 190 user sessions per hour for an eight hour period.

Collecting sample data. If you don't know the level of FOCUS usage at your site, and cannot apply the above technique, you may just wish to collect data for a given period using a large log file and see how long it takes to fill or how much is used. For example, allocate a 100-cylinder log file for use in production for one business day and at day's end, use ISPF to determine how much of the log was populated. If you used 10 cylinders of the log file on a fairly typical day, a good estimate for the size is 10 cylinders per day. Extend this to determine numbers of cylinders required for longer periods. The only penalty for under-sizing your log is that logging stops when the log is full.

In designing the log file, also take the following considerations into account:

- ❑ You can easily FTP or email small logs to other locations (for example, your Desktop) for analysis with tools such as WebFOCUS or Excel.
- ❑ Large log files can capture data for longer periods without continuous monitoring, but require more time for usage review and are slower when generating reports.

Activating FOCLOG

Example:

Requesting a Default Form of the Log on z/OS

Requesting a Default Form of the Log on z/VM

Requesting a Detailed Form of the Log on z/OS

Requesting a Detailed Form of the Log on z/VM

Requesting a Session Summary Form of the Log on z/OS

Requesting a Session Summary Form of the Log on z/VM

Reference:

Notes on Activation and Deactivation of Logging

Usage can be logged at three levels, producing either a short form log, a detailed log, or a summary log, as described in the examples that follow.

Activation of logging requires three components:

1. On z/OS, the presence of a FOCUSLOG member in the concatenation of data sets allocated to DDNAME ERRORS. On z/VM, the existence of a FOCUSLOG ERRORS file on a disk accessed by FOCUS users.
2. The FOCUSLOG file must contain the name of the FOCLOG log file.
3. An indicator for the type of log desired.

Example: Requesting a Default Form of the Log on z/OS

The FOCUSLOG file in this example points to a FOCLOG file named FLHLQ.FOCLOG.DATA. Because the data set name is followed by blanks, this generates a default form of the log containing information about all attributes defined in the FOCLOG Master File, except the data set names of the FOCEXEC files, Access Files, and Master Files.

```
FLHLQ.FOCLOG.DATA
```

Example: Requesting a Default Form of the Log on z/VM

The FOCUSLOG file in this example points to an SFS directory named VMSYSU:FOCLOG.DATA. Because the directory name is followed by blanks, this generates a default form of the log containing information about all attributes defined in the FOCLOG Master File, except the data set names of the FOCEXEC files, Access Files, and Master Files:

```
VMSYSU:FOCLOG.DATA
```

Example: Requesting a Detailed Form of the Log on z/OS

This FOCUSLOG member produces a detailed form of the log containing everything in the FOCLOG Master File plus the data set names of FOCEXEC files, Master Files, and Access Files.

To create the detailed log, the FOCLOG file named FLHLQ.FOCLOG.DATA must be followed by at least one blank and the word *DETAIL*:

```
FLHLQ.FOCLOG.DATA DETAIL
```

Example: Requesting a Detailed Form of the Log on z/VM

The FOCUSLOG file in this example points to an SFS directory named VMSYSU:FOCLOG.DATA. Because the directory name is followed by the keyword DETAIL, this generates a detailed form of the log containing everything in the FOCLOG Master File plus the data set names of FOCEXEC files, Master Files, and Access Files:

```
VMSYSU:FOCLOG.DATA DETAIL
```

Example: Requesting a Session Summary Form of the Log on z/OS

This FOCUSLOG member points to the same log file as the one defined in *Requesting a Default Form of the Log on z/OS* on page 358, but this request creates a session summary form of the log containing only information contained in the session segment of the FOCLOG Master File.

To create a session summary log, the FOCLOG file named FLHLQ.FOCLOG.DATA must be followed by at least one blank plus the word *SESSION*:

```
FLHLQ.FOCLOG.DATA SUMMARY
```

Example: Requesting a Session Summary Form of the Log on z/VM

The FOCUSLOG file in this example points to an SFS directory named VMSYSU:FOCLOG.DATA. Because the directory name is followed by the keyword SUMMARY, this generates a session summary form of the log containing only information contained in the session segment of the FOCLOG Master File.

```
VMSYSU:FOCLOG.DATA SUMMARY
```

Reference: Notes on Activation and Deactivation of Logging

When FOCUS is initiated, it checks for the presence of the log file. If the FOCUSLOG file is present and contains the name of a log file, usage data is written to memory during the FOCUS session. The information stored in memory is written to the log file at FIN, during FOCUS termination. Therefore, FOCLOG does not produce any unnecessary overhead or use additional CPU cycles for capturing usage data.

Because logging depends on having a FOCUSLOG file that points to a log file, you can deactivate FOCLOG by taking any of the following actions. These actions are listed in the recommended order:

- Rename the log file itself so that the FOCUSLOG file no longer points to a valid log file.

  You can also use this technique to save an old log and start a new log file. For example, if the old log file is named FLHLQ.FOCLOG.DATA, rename it to FLHLQ.FOCLOG.DATA1, and allocate a new file as FLHLQ.FOCLOG.DATA. **Note:** Log replacement should be done as quickly as possible to avoid losing data that should be captured in the log.

- Rename the pointer to the log file in your FOCUSLOG member. For example, assume the FOCUSLOG file contains the following log file name:

  On z/OS:

  ```
  FLHLQ.FOCLOG.DATA
  ```

  On z/VM:

  ```
  VMSYSU:FOCLOG.DATA
  ```

  Change this entry in FOCUSLOG so that it no longer points to a valid log file or SFS directory. For example, if no file exists with the name FOCLOG.DATAOFF, change the pointer in FOCUSLOG to reference this non-existent file:

  On z/OS:

  ```
  FLHLQ.FOCLOG.DATAOFF
  ```

  On z/VM:

  ```
  VMSYSU:FOCLOG.DATAOFF
  ```

- Rename the FOCUSLOG file so that it is not found under the name FOCUSLOG. For example, change the name to FOCLGOFF.

Validating the FOCLOG Configuration on z/OS

Example:

Sample FOCUS Session on z/OS

This step validates your FOCLOG configuration and enables you to confirm that the log file was allocated and defined properly. It does not test FOCUS functionality, which is not impacted by FOCLOG.

1. To create the validation environment, make a test copy of your FOCUS production CLIST or batch job. This CLIST should allocate the production versions of the ERRORS, MASTER, and FOCEXEC DDNAMEs.
2. Run the test CLIST or batch job to enter FOCUS.
3. Next, execute a request to populate the log:

   At the FOCUS prompt, issue the following command and press Enter to create a temporary FOCUS database named CAR and load it with data:

   ```
   EX FLVALPOP
   ```

   The following messages display:

   ```
   >  NEW FILE CAR      ON 09/15/2007 AT 09.06.57
    CAR      ON 09/15/2007 AT 09.06.57
    WARNING..TRANSACTIONS ARE NOT IN SAME SORT ORDER AS FOCUS FILE
    PROCESSING EFFICIENCY MAY BE DEGRADED
    TRANSACTIONS:         TOTAL =    53  ACCEPTED=    53  REJECTED=     0
    SEGMENTS:             INPUT =   102  UPDATED =     0  DELETED =     0
   ```

   You can ignore any warning messages. The important thing to note is that 53 transactions were accepted and that 102 segments were input. **Note:** If you have problems running this procedure, please contact the Information Builders Customer Support Services staff.
4. Exit from FOCUS by issuing the following command and pressing Enter:

   ```
   FIN
   ```

   Ending the FOCUS session updates the log with the information about the procedure you executed.
5. Run the test CLIST or batch job to enter FOCUS again.

6. You will now execute a request to produce a validation report from the log.

a. Issue the following command to allocate your log file so that you can issue a report request against it. Replace *flhlq* with the high level qualifier for your log file data set. (**Note:** Before pressing Enter, make sure the data set name is the name of the log file you specified in the FOCUSLOG member of your production ERRORS.DATA library.):

```
DYNAM ALLOC DD FOCLOG DA flhlq.FOCLOG.DATA SHR REU
```

b. Issue the following command and then press Enter to report from the log:

```
EX FLVALRPT
```

The following messages display to indicate that the report is ready to view:

```
>   NUMBER OF RECORDS IN TABLE=        3  LINES=      3
     PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
```

Press Enter to view the report. It should be similar to the following, but display your session statistics and user ID:

```
PAGE     1

VALIDATION OF SESSION COMMANDS ISSUED

STARTDATE   SESSTART      USERID   COMMAND  FNAME     RECORDS    LINES
---------   ---------     ------   -------  -----     -------    -----
2007/09/15  11:45:05.260  FLHLQ    CREATE   .               0        0
                          FLHLQ    MODIFY   CAR           102        0
                          FLHLQ    TABLE    CAR            18        1
```

This output shows that the log is working properly. After producing the report, the procedure deallocates DDNAMEs CAR and FOCLOG. **Note:** If you have problems running this request, please contact the Information Builders Customer Support Services staff.

Press Enter to close the report window.

7. Exit from FOCUS by issuing the following command and then pressing Enter:

```
FIN
```

You have now completed the configuration and validation process. You can now notify your FOCUS Administrator that the product is available for use.

Note: This validation procedure added data to the log file that does not reflect actual FOCUS usage at your site. Therefore, you should edit the log file to delete this data before putting the log into production. To delete the data, open the log file in the ISPF Edit Panel and delete the lines that contain the data.

Example: Sample FOCUS Session on z/OS

When you enter an online FOCUS session, you see a banner similar to the following. The two carets at the bottom (> >) are the FOCUS prompt, although your site may have changed the prompt:

```
  FOCUS  7.6.3   09/15/2007  10.12.52

OBSERVED CPU:
******* CEC:  machine type N/A   model ID N/A              capacity N/A
******* LPAR: name N/A                                     capacity N/A
******* VM:   name N/A                                     capacity N/A
******* Processor AF4A Model 2066-00 Max 02  Site
LICENSED CPU(S):
******* Processor 5394 Model 9672-F0 Max 01 >Registration 57E3C86A0887
```

The following command executes the request that populates the log:

```
>  > ex flvalpop
```

The following messages display:

```
>   NEW FILE CAR      ON 09/15/2007 AT 09.06.57
 CAR      ON 09/15/2007 AT 09.06.57
 WARNING..TRANSACTIONS ARE NOT IN SAME SORT ORDER AS FOCUS FILE
 PROCESSING EFFICIENCY MAY BE DEGRADED
 TRANSACTIONS:         TOTAL =    53  ACCEPTED=    53  REJECTED=     0
 SEGMENTS:             INPUT =   102  UPDATED =     0  DELETED =     0
```

The next command ends the FOCUS session and updates the log with statistics regarding the FLVALPOP request:

```
> fin
```

Next, we run the CLIST again to go back into FOCUS:

```
  FOCUS  7.6.3   09/15/2007  10.12.52

OBSERVED CPU:
******* CEC:  machine type N/A   model ID N/A              capacity N/A
******* LPAR: name N/A                                     capacity N/A
******* VM:   name N/A                                     capacity N/A
******* Processor AF4A Model 2066-00 Max 02  Site
LICENSED CPU(S):
******* Processor 5394 Model 9672-F0 Max 01 >Registration 57E3C86A0887
```

Now we allocate the log file and issue the FLVALRPT request to view information from the log:

```
>  > dynam alloc dd foclog da flhlq.foclog.data shr reu
>  > ex flvalrpt
```

The following messages display to indicate that the report is ready to view

```
>   NUMBER OF RECORDS IN TABLE=         3  LINES=      3

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
```

Pressing Enter displays the report:

```
PAGE     1

VALIDATION OF SESSION COMMANDS ISSUED

STARTDATE   SESSTART      USERID    COMMAND   FNAME     RECORDS    LINES
---------   ---------     ------    -------   -----     -------    -----
2007/09/15  11:45:05.260  FLHLQ     CREATE    .               0        0
                          FLHLQ     MODIFY    CAR           102        0
                          FLHLQ     TABLE     CAR            18        1
```

Next, press Enter to close the report window and return to the FOCUS prompt.

Execute the FIN command to end the FOCUS session:

```
> fin
```

Now empty the log file (using ISPF Edit) before putting the log into production.

Validating the FOCLOG Configuration on z/VM

Example:

Sample FOCUS Session on z/VM

This step validates your FOCLOG configuration and enables you to confirm that the log file was accessed and defined properly. It does not test FOCUS functionality, which is not impacted by FOCLOG.

1. To create the validation environment, make a test copy of the EXEC that your customers use to access FOCUS. This EXEC should FILEDEF the production versions of the ERRORS, MASTER, and FOCEXEC files.
2. Run the test EXEC to enter FOCUS.

3. Next, execute a request to populate the log:

   Edit the FLVALPOP FOCEXEC and delete the following lines near the top of the file, then save the edited file:

   ```
   00006 -? TSO DDNAME CAR
   00007 -IF &DSNAME EQ ' ' THEN GOTO ALLOC1
   00008 DYNAM FREE DD CAR
   00009 -ALLOC1
   00010 DYNAM ALLOC DD CAR  RECFM F LRECL 4096 BLKSIZE 4096 -
   00011       DSORG PS                                     -
   00012       SPACE 1,3 CYL
   00013 -RUN
   ```

   At the FOCUS prompt, issue the following command and press Enter to create a temporary FOCUS database named CAR and load it with data:

   ```
   EX FLVALPOP
   ```

   The following messages display:

   ```
   >  NEW FILE CAR     FOCUS   A1      ON 09/15/2007 AT 09.06.57
    CAR     FOCUS   A1      ON 09/15/2007 AT 09.06.57
    WARNING..TRANSACTIONS ARE NOT IN SAME SORT ORDER AS FOCUS FILE
    PROCESSING EFFICIENCY MAY BE DEGRADED
    TRANSACTIONS:         TOTAL =    53  ACCEPTED=    53  REJECTED=     0
    SEGMENTS:             INPUT =   102  UPDATED =     0  DELETED =     0
   ```

   You can ignore any warning messages. The important thing to note is that 53 transactions were accepted and that 102 segments were input. **Note:** If you have problems running this procedure, please contact the Information Builders Customer Support Services staff.

4. Exit from FOCUS by issuing the following command and pressing Enter:

   ```
   FIN
   ```

   Ending the FOCUS session updates the log with the information about the procedure you executed.

5. Run the test EXEC to enter FOCUS again.

6. You will now execute a request to produce a validation report from the log.

a. Issue the following commands to ACCESS and FILEDEF your log file so that you can issue a report request against it. (**Note:** Before pressing Enter, make sure the data set name is the name of the log file you specified in the FOCUSLOG file.). For example:

```
ACCESS VMSYSU:USER1.FOCLOG B
CMS FILEDEF FOCLOG DISK FOCLOG DATA B
```

b. Issue the following command and then press Enter to report from the log:

```
EX FLVALRPT
```

The following messages display to indicate that the report is ready to view:

```
>   NUMBER OF RECORDS IN TABLE=        3  LINES=      3
     PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
```

Press Enter to view the report. It should be similar to the following, but display your session statistics and user ID:

```
PAGE     1

VALIDATION OF SESSION COMMANDS ISSUED

STARTDATE   SESSTART      USERID    COMMAND   FNAME     RECORDS    LINES
---------   ---------     ------    -------   -----     -------    -----
2007/09/15  11:45:05.260  FLHLQ     CREATE    .               0        0
                          FLHLQ     MODIFY    CAR           102        0
                          FLHLQ     TABLE     CAR            18        1
```

This output shows that the log is working properly. **Note:** If you have problems running this request, please contact the Information Builders Customer Support Services staff.

Press Enter to close the report window.

7. Exit from FOCUS by issuing the following command and then pressing Enter:

```
FIN
```

You have now completed the configuration and validation process. You can now notify your FOCUS Administrator that the product is available for use.

Note: This validation procedure added data to the log file that does not reflect actual FOCUS usage at your site. Therefore, you should edit the log file to delete this data before putting the log into production. To delete the data, open the log file in XEDIT and delete the lines that contain the data.

Example: Sample FOCUS Session on z/VM

When you enter an online FOCUS session, you see a banner similar to the following. The two carets at the bottom (> >) are the FOCUS prompt, although your site may have changed the prompt:

```
FOCUS  7.6.3   11/02/2007  09.50.12  GEN231
OBSERVED CPU:
****** Processor 00AF5A Model 2066-000 Max 001 >Site GEN2
LICENSED CPU(S):
****** Processor 005394 Model 9672-0F0 Max 001 >Registration 57E3C86A0887
```

The following command executes the request that populates the log:

```
>  > ex flvalpop
```

The following messages display:

```
NEW FILE CAR     FOCUS   A1 ON 11/02/2007 AT 09.55.09
 CARFL   FOCUS   A1 ON 11/02/2007 AT 09.55.11
 WARNING..TRANSACTIONS ARE NOT IN SAME SORT ORDER AS FOCUS FILE
 PROCESSING EFFICIENCY MAY BE DEGRADED
 TRANSACTIONS:         TOTAL =    53  ACCEPTED=    53  REJECTED=     0
 SEGMENTS:             INPUT =   102  UPDATED =     0  DELETED =     0
```

The next command ends the FOCUS session and updates the log with statistics regarding the FLVALPOP request:

```
> fin
```

Next, we ACCESS and FILEDEF the log file:

```
ACCESS VMSYSU:USER1.FOCLOG B
CMS FILEDEF FOCLOG DISK FOCLOG DATA B
```

Then, we go back into FOCUS:

```
FOCUS  7.6.3   11/02/2007  09.55.10  GEN231
OBSERVED CPU:
****** Processor 00AF5A Model 2066-000 Max 001 >Site GEN2
LICENSED CPU(S):
****** Processor 005394 Model 9672-0F0 Max 001 >Registration 57E3C86A0887
```

Now we issue the FLVALRPT request to view information from the log:

```
>  > ex flvalrpt
```

The following messages display to indicate that the report is ready to view

```
>   NUMBER OF RECORDS IN TABLE=        3  LINES=      3

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
```

Pressing Enter displays the report:

```
PAGE     1

VALIDATION OF SESSION COMMANDS ISSUED

STARTDATE   SESSTART      USERID    COMMAND   FNAME     RECORDS    LINES
---------   --------      ------    -------   -----     -------    -----
2007/09/15  11:45:05.260  FLHLQ     CREATE    .               0        0
                          FLHLQ     MODIFY    CAR           102        0
                          FLHLQ     TABLE     CAR            18        1
```

Next, press Enter to close the report window and return to the FOCUS prompt.

Execute the FIN command to end the FOCUS session:

```
> fin
```

Now empty the log file (using XEDIT) before putting the log into production.

Running the FOCLOG Reports

FOCLOG is now configured. See *FOCLOG Reporting* on page 375, for instructions about reporting on the captured data.

Information Captured in the FOCLOG File

Reference:

Session Segment Information

Command Segment Information

Master Segment Information

Capturing the Access File Data Set Name

Data Set Segment Information

Master File Structure Diagram

The FOCLOG file is a fixed format physical sequential file. FOCLOG is distributed with a Master File that describes the FOCLOG file and enables you to report from it.

The Master File has four segments:

- ❑ The session segment provides information specific to a user's session.
- ❑ The command segment provides information about FOCUS commands issued during a user's session.
- ❑ The master segment provides Master File names, file types, and the ID of the sink machine on which each Master File resides.
- ❑ The data set name segment provides names and locations of data sets used during a user's session.

Reference: Session Segment Information

Field Name	Description
`RECTYPE`	Session segment record type (SESS) (format A4)
`STARTDT`	Internal date value; see DEFINE field named STARTDATE for display value (format A6)
`MICROSEC`	Microseconds since Jan. 1, 2000 (format A17)
`SESSTART`	Local time - HH:MM:SS.NN. Captured down to the nanosecond, ensuring unique session start times for users. (format A12)
`USERID`	User Identifier; user ID or batch job name. (format A8)
`SESDURATION`	Session duration, seconds. SESDURATION is displayed with 6 digits, a decimal point, then 6 digits. (format A17)

Field Name	Description
SESCPU	Session CPU time, seconds. SESCPU is displayed with 3 digits, a decimal point, then 3 digits. (format A17)
SESEXCP	Session total EXCP (format A8)
FOCREL	Focus release (format A16)
SITECODE	IBI sitecode (format A8)
MSO	Flag - 'Y' if MSO application (format A1)
BATCH	Flag - 'Y' if BATCH application (format A1)
CPUID	CPU INFORMATION (format A12)
OPSYS	Operating system (format A8)
OPSYSREL	OS version/release (format A8)
LPARNAM	Logical Partition Name (format A8)
IBIREG	IBI registration flag - 'Y' if IBI registration (format A1)
JOBNAME	z/OS job name (format A8)
JOBID	z/OS job ID (format A8)
MODEL_ID	Processor Model ID (format A8)
BASEDATE	DEFINE: Base date constant for date calculations (1900 DEC 31) (format YYMD)
STARTDATE	DEFINE: Start date for display (format YYMD)
STARTMONTH	DEFINE: Month of session start (format MTR)
STARTQTR	DEFINE: Quarter of session start (format YYQ)
STARTYYMTR	DEFINE: Start date with month translated (YYMTR)
STARTWEEK	DEFINE: Date of beginning of week of session start (format YYMD) **Note:** This DEFINE must be commented out when running the FOCLOG Release 1.3 version with FOCUS releases 6.0 or 6.8.
STARTHOUR	DEFINE: Hour of session start (format A2)
SERIAL_NUM	DEFINE: CPU ID serial number (format A6)

Field Name	Description
MODEL_NUM	DEFINE: CPU ID model number (format A4)
PROCESSOR_ID	DEFINE: PROCESSOR_ID combines output from the MODEL_ID and MODEL_NUM fields (A8)

Reference: Command Segment Information

Field Name	Description
RECTYPE	Command segment record type (CMDS) (format A4)
COMMAND	FOCUS command (e.g. TABLE, MODIFY) (format A8)
FOCEXEC	FOCEXEC name if run by an EXEC command (format A8)
LINENUM	Line number of command in the FOCEXEC (format A6)
CMDSTART	Seconds since start of session. CMDSTART is displayed with 5 digits, a decimal point, then 3 digits. (format A17)
CMDCPU	CPU time to execute the command. CMDCPU is displayed with 5 digits, a decimal point, then 3 digits (format A17)
CMDDURATION	Duration of command, in seconds. CMDDURATION is displayed with 5 digits, a decimal point, then 3 digits (format A17)
BASEIO	Statistics of last command issued: Baseio (format A7)
SORTIO	Statistics of last command issued: Sortio (format A7)
RECORDS	Statistics of last command issued: Records for Table, input for Modify (format A7)
LINES	Statistics of last command issued: Lines for TABLE, changed for MODIFY (format A7)
READS	Statistics of last command issued: Reads for TABLE, deleted for MODIFY (format A7)
CMDEXCP	Statistics of last command issued: EXCPs for command (format A7)
HLRECL	Hold file lrecl (format A5)
POOLED	Flag - 'Y' if pooled tables used (format A1)

Field Name	Description
EXTSORT	Flag - 'Y' if external sort used (format A1)
OUTFLAG	Output format numeric value (internal value; see DEFINE field OUTPUT_TYPE for display value) (format A3)
OFFLINE	Flag - 'Y' if OFFLINE output (format A1)
FOCEXECDSN	FOCEXEC file library name - populated in long log format only. (format A44)
OUTPUT_TYPE	DEFINE: output type name (format A8). Decode OUTFLAG table
OUTVOLUME	DEFINE: calculated file size in bytes (format D15). IF OUTPUT_TYPE is HOLD, SAVE, SAVB, or PCHOLD THEN HLRECL x LINES ELSE 0.
FOCEXEC	DEFINE: IF FOCEXEC LT 'A' THEN ' ', ELSE FOCEXEC (format A8)

Reference: Master Segment Information

Field Name	Description
RECTYPE	Master segment record type (MASS) (format A4)
FNAME	Master File name (format A8)
SUFF	File suffix (FOC, FIX, etc.) (format A8)
SINKID	Sink identifier (if SU used, otherwise blank) (format A8)
MASTERDSN	Master file library name - populated in long log format only. (format A44)
ACCESSDSN	Access file library name - populated in long log format only. (format A44). For information about capturing ACCESSDSN, see *Capturing the Access File Data Set Name* on page 373.
FILE_TYPE	DEFINE: Translates SUFF into a more understandable name (format A8)

Reference: Capturing the Access File Data Set Name

If you are using an Access File DDNAME other than ACCESS, you may be able to allocate those data sets to DDNAME ACCESS instead in order to capture the Access File data set names in the log file.

Reference: Data Set Segment Information

Field Name	Description
`RECTYPE`	Data set name record type (DSNS) (format A4)
`DDNAME`	z/OS DDNAME (format A8)
`DSNAME`	Data set name (format A44)
`DDEXCP`	EXCPs for ddname (format A7)
`DSNORD`	Order number of concatenation (format A3)
`DEVFLAG`	Flag - 'Y' if possible missing EXCPs (format A1)
`DSNTEMP`	Flag - 'Y' if dsn is a temp file (format A1)

Reference: Master File Structure Diagram

The following diagram shows the structure of the FOCLOG Master File and the relationships between its segments. It lists the first four fields in each segment:

```
NUMBER OF ERRORS=     0
NUMBER OF SEGMENTS=   4  ( REAL=    4  VIRTUAL=   0 )
NUMBER OF FIELDS=    55  INDEXES=   0  FILES=     1
NUMBER OF DEFINES=    7
TOTAL LENGTH OF ALL FIELDS=  509
SECTION 01
             STRUCTURE OF FIX      FILE FOCLOG   ON 05/07/04 AT 13.23.43

         SESS
 01      S0
**************
*RECTYPE     **
*STARTDT     **
*MICROSEC    **
*SESSTART    **
*            **
***************
 **************
       I
       +-----------------+
       I                 I
       I CMD             I DSN
 02    I N         04    I N
**************    **************
*RECTYPE     **   *RECTYPE     **
*COMMAND     **   *DDNAME      **
*FOCEXEC     **   *DSNAME      **
*LINENUM     **   *DDEXCP      **
*            **   *            **
***************   ***************
 **************    **************
       I
       I
       I
       I MAS
 03    I N
**************
*RECTYPE     **
*FNAME       **
*SUFF        **
*SINKID      **
*            **
***************
 **************
```

FOCLOG Reporting

FOCLOG comes packaged with a Master File that gives FOCUS access to the log file. It also comes with a menu driven interface with usage reports that you can start running as soon as you decide you have collected enough usage data.

Using the Menu-Driven FOCLOG Reporting Interface

The menu-driven FOCLOG reporting interface provides a review of usage patterns and detailed analysis of the FOCLOG file. Overlap between the summary and detailed report contents support easy analysis of the usage data. Run-time options enable easy adjustment of the collection and aggregation periods and sort criteria to aid in analyzing apparent anomalies.

The reports are organized into dimensions that group similar types of reports and concepts of analysis.

Reference: Specifying Values Used in Report Generation

Before running the reporting interface, you must edit the FLPROF FOCEXEC file to specify the company name you want to display in the report headings and to allocate or FILEDEF your FOCLOG log file.

The following is a listing of the FLPROF FOCEXEC:

```
-***************************************************************************
-*
-* INFORMATION BUILDERS INC.
-* FOCLOG STATISTICAL REPORTS
-*
-* PROFILE FOR FOCLOG REPORTING
-*                                                       @MFSM_NOPROLOG@
-***************************************************************************


-* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-* PUT YOUR COMPANY NAME HERE AS IT SHOULD APPEAR ON ALL REPORTS.
-SET &&COMPANY   = 'INFORMATION BUILDERS INC.';
-*
-IF &FOCMODE EQ 'TSO' OR 'MVS' GOTO MVS_ALLOC ;
-*                 VM/CMS USERS
-* CHANGE THE FOLLOWING LINE TO REFLECT THE SFS DIRECTORY NAME
-*        YOU CHOSE FOR YOUR FOCLOG FILE.
CMS FILEDEF FOCLOG DISK VMSYSU:FOCLOG.DATA
-GOTO CONT
-MVS_ALLOC
-*                MVS/TSO USERS
-* CHANGE THE FOLLOWING LINE TO REFLECT THE DATASET NAME
-*        YOU CHOSE FOR YOUR FOCLOG FILE.
DYNAM ALLOC FILE FOCLOG DA FLHLQ.FOCLOG.DATA SHR REUSE
-RUN
-CONT
-* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Make the following changes:

- ❑ Replace ‘INFORMATION BUILDERS, INC.’ with your company name in the following line:

  ```
  -SET &&COMPANY   = 'INFORMATION BUILDERS INC.';
  ```

- ❑ If you are running on z/VM, replace VMSYSU:FOCLOG.DATA with your SFS directory name in the following line:

  ```
  CMS FILEDEF FOCLOG DISK VMSYSU:FOCLOG.DATA
  ```

- ❑ If you are running on z/OS, replace FLHLQ.FOCLOG.DATA with the fully qualified name of your FOCLOG log file in the following line:

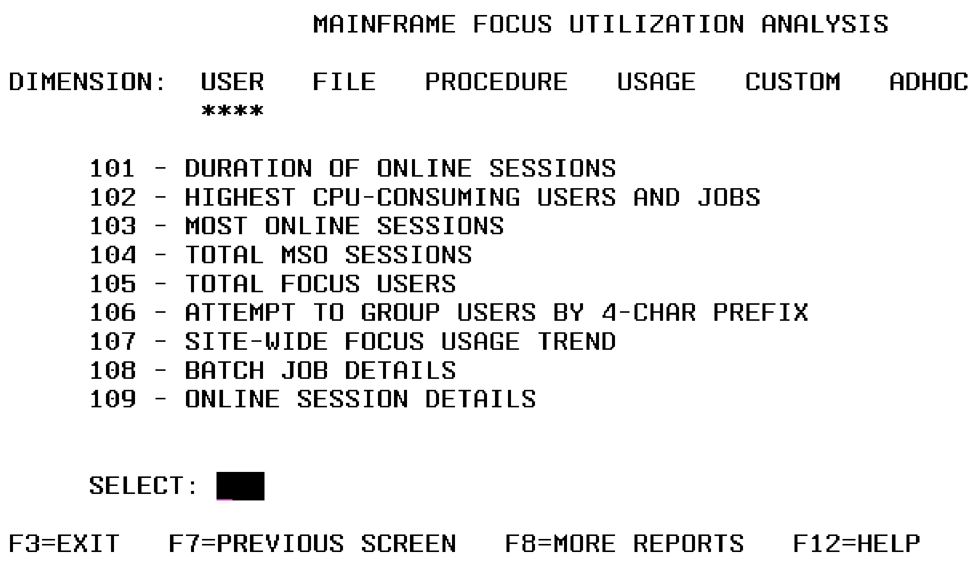
  ```
  DYNAM ALLOC FILE FOCLOG DA FLHLQ.FOCLOG.DATA SHR REUSE
  ```

Save the edited version of FLPROF FOCEXEC before running the reports.

Procedure: How to Invoke the FOCLOG Reporting Interface

From FOCUS, issue the following command to display report selection menu:

```
EX FLMENU
```



```
                    MAINFRAME FOCUS UTILIZATION ANALYSIS

 DIMENSION:  USER   FILE   PROCEDURE   USAGE   CUSTOM   ADHOC
             ****

      101 - DURATION OF ONLINE SESSIONS
      102 - HIGHEST CPU-CONSUMING USERS AND JOBS
      103 - MOST ONLINE SESSIONS
      104 - TOTAL MSO SESSIONS
      105 - TOTAL FOCUS USERS
      106 - ATTEMPT TO GROUP USERS BY 4-CHAR PREFIX
      107 - SITE-WIDE FOCUS USAGE TREND
      108 - BATCH JOB DETAILS
      109 - ONLINE SESSION DETAILS


      SELECT:

 F3=EXIT   F7=PREVIOUS SCREEN   F8=MORE REPORTS   F12=HELP
```

Use the PF8 and PF7 keys to move forward and backward through the report selection menus for the following five reporting dimensions listed at the top of the screen:

- ❑ USER. These reports describe user-oriented activity on the system.
- ❑ FILE. These reports describe files accessed by FOCUS.
- ❑ PROCEDURE. These reports describe FOCUS programs running on the system.
- ❑ USAGE. These reports describe FOCUS activity.
- ❑ CUSTOM. These reports are provided by the user. For information on creating your own reports and adding them to the reporting interface, see *How to Build and Catalog a Custom FOCLOG Report* on page 381.

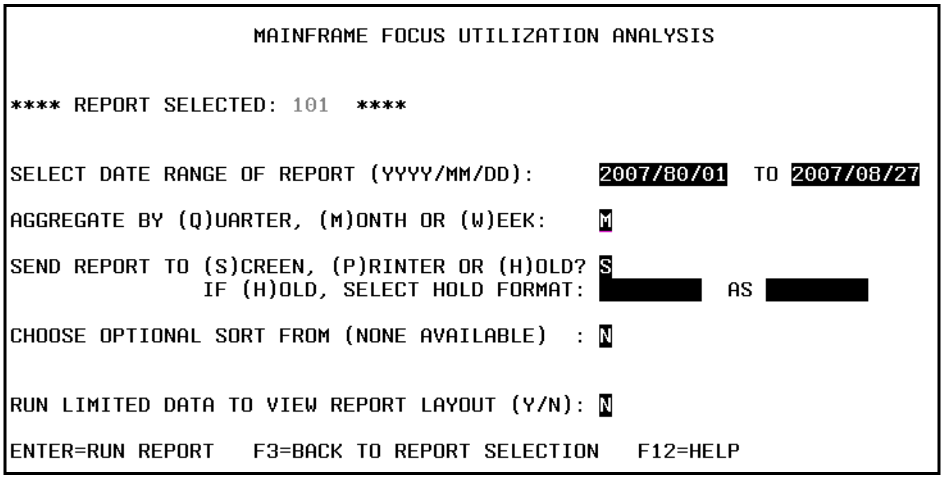
To select a specific report, enter its three-digit report number. If you know a report's three-digit number, you can enter that number from any screen.

Tip: If you are on a screen and want to run a report listed on that screen, you can enter the one or two-digit number to the right of the leftmost digit (which is the screen number). For example, if you are on screen 1, you can run report 109 by entering 9 or 09.

If you want to enter FOCUS from the interface in order to run your own requests against the FOCLOG file, select the ADHOC dimension.

Use the function keys as described at the bottom of each screen. Look for helpful messages that appear below the function key line.

Once you have selected a report, an options screen displays:



```
                    MAINFRAME FOCUS UTILIZATION ANALYSIS

**** REPORT SELECTED: 101   ****

SELECT DATE RANGE OF REPORT (YYYY/MM/DD):     2007/80/01  TO 2007/08/27

AGGREGATE BY (Q)UARTER, (M)ONTH OR (W)EEK:    M

SEND REPORT TO (S)CREEN, (P)RINTER OR (H)OLD? S
               IF (H)OLD, SELECT HOLD FORMAT:           AS

CHOOSE OPTIONAL SORT FROM (NONE AVAILABLE)  : N


RUN LIMITED DATA TO VIEW REPORT LAYOUT (Y/N): N

ENTER=RUN REPORT   F3=BACK TO REPORT SELECTION   F12=HELP
```

You can specify the following options on this screen:

- ❏ The date range covered. Supply the dates in YYYY/MM/DD format, using the default values as a guide. The default is a three-month range from the current date backwards.
- ❏ The aggregation period (Quarter, Month, Week). This applies to reports that have a calendar sort level and can accommodate the selection.
- ❏ The report destination (Screen, Print, Hold file). Enter:

  S to see the report on your screen.

  P to send the report to your printer.

  H to store the report results in a HOLD file. By default, the format type will be a BINARY HOLD file. You can specify a different format and an AS phrase to supply a name other than the default name HOLD.

- ❑ Sort options, if any are available for that report. The first sort is generally the time period. Some reports can be additionally sorted on other columns, such as duration, CPU, EXCP, records, and lines. Only the sorts applicable to that report are shown. If none are available, NONE AVAILABLE displays. Use the first letter of the word to select that sort column, or, if you do not want an additional sort level, either leave it blank or enter N (None).
- ❑ The option of making a limited request to review report layout. Enter Y at this prompt to see a version of the report with limited data. A report may not display at all if no data matches the criteria for selecting the limited records.

The date period you select for the first request in a session remains in effect until you change it, which you can do on any report selection screen. This is also true for aggregation periods and report destinations.

Example: Running a FOCLOG Report

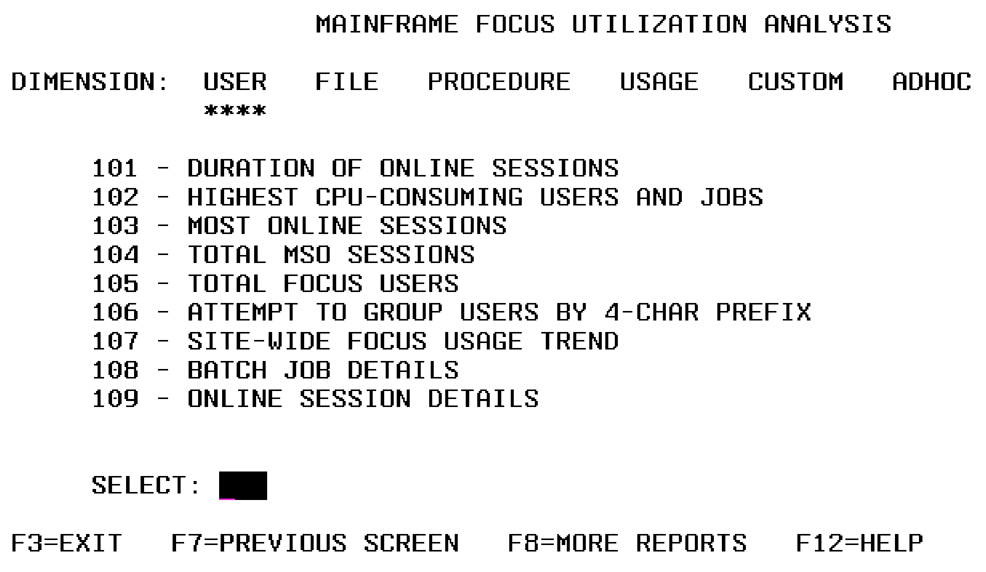
The following example runs FOCLOG report FLRPT204.

First, make sure you have access to the FOCLOG log file and that you added the appropriate DYNAM or FILEDEF command in the FLPROF FOCEXEC file.

Then, issue the following command to execute the reporting interface:

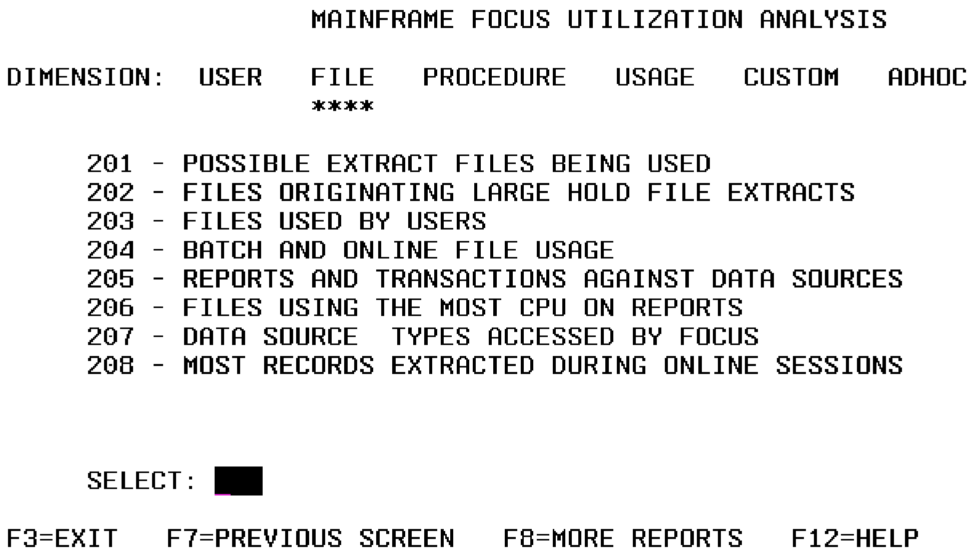
```
ex flmenu
```

The following screen displays:



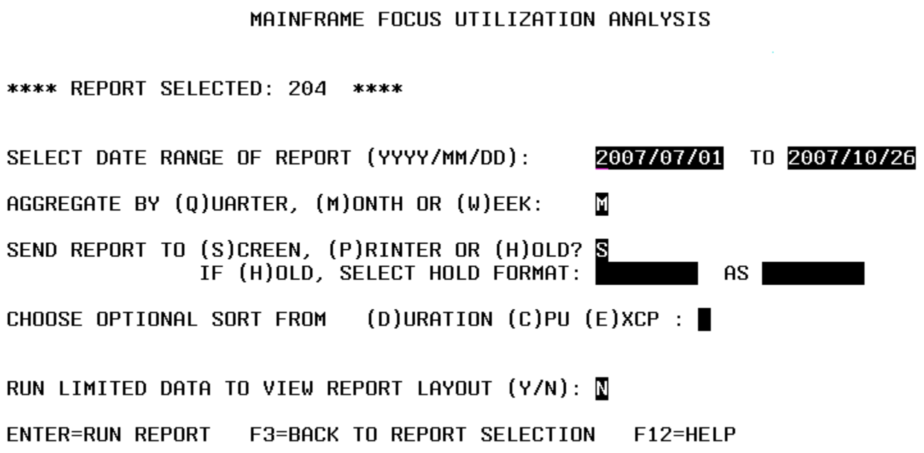
```
                   MAINFRAME FOCUS UTILIZATION ANALYSIS

DIMENSION:  USER   FILE   PROCEDURE   USAGE   CUSTOM   ADHOC
            ****

     101 - DURATION OF ONLINE SESSIONS
     102 - HIGHEST CPU-CONSUMING USERS AND JOBS
     103 - MOST ONLINE SESSIONS
     104 - TOTAL MSO SESSIONS
     105 - TOTAL FOCUS USERS
     106 - ATTEMPT TO GROUP USERS BY 4-CHAR PREFIX
     107 - SITE-WIDE FOCUS USAGE TREND
     108 - BATCH JOB DETAILS
     109 - ONLINE SESSION DETAILS


     SELECT:

F3=EXIT   F7=PREVIOUS SCREEN   F8=MORE REPORTS   F12=HELP
```

Press F8 to open the FILE screen:



```
                  MAINFRAME FOCUS UTILIZATION ANALYSIS

DIMENSION:  USER   FILE   PROCEDURE   USAGE   CUSTOM   ADHOC
                   ****

     201 - POSSIBLE EXTRACT FILES BEING USED
     202 - FILES ORIGINATING LARGE HOLD FILE EXTRACTS
     203 - FILES USED BY USERS
     204 - BATCH AND ONLINE FILE USAGE
     205 - REPORTS AND TRANSACTIONS AGAINST DATA SOURCES
     206 - FILES USING THE MOST CPU ON REPORTS
     207 - DATA SOURCE  TYPES ACCESSED BY FOCUS
     208 - MOST RECORDS EXTRACTED DURING ONLINE SESSIONS



     SELECT:

F3=EXIT   F7=PREVIOUS SCREEN   F8=MORE REPORTS   F12=HELP
```

Enter the number 204 in the SELECT field to run FLRPT204. The Options screen opens:



```
                      MAINFRAME FOCUS UTILIZATION ANALYSIS

**** REPORT SELECTED: 204  ****

SELECT DATE RANGE OF REPORT (YYYY/MM/DD):     2007/07/01  TO 2007/10/26

AGGREGATE BY (Q)UARTER, (M)ONTH OR (W)EEK:    M

SEND REPORT TO (S)CREEN, (P)RINTER OR (H)OLD? S
               IF (H)OLD, SELECT HOLD FORMAT:            AS

CHOOSE OPTIONAL SORT FROM   (D)URATION (C)PU (E)XCP :


RUN LIMITED DATA TO VIEW REPORT LAYOUT (Y/N): N

ENTER=RUN REPORT   F3=BACK TO REPORT SELECTION   F12=HELP
```

Accept the default options and run the report by pressing Enter. A report similar to the following displays:

```
                  MAINFRAME FOCUS UTILIZATION ANALYSIS
                       INFORMATION BUILDERS INC.
                JULY  1, 2007 -      OCTOBER 26, 2007
                      BATCH AND ONLINE FILE USAGE


BATCH/ SESSION                    DURATION   CPU USAGE
ONLINE  COUNT  FILE TYPE          HH:MM:SS    HH:MM:SS   EXCPS/1000
------ ------- --------- ----------- -----------  ----------
BATCH        2                       07:28       00:05           0
               FOCUS               1:58:47       01:06           0

TOTAL                              2:06:16       01:11           0
------------------------------------------------------------
ONLINE       8                       04:14       00:07           0
               DB2                   00:00       00:00           0
               FIXED               2:29:16       00:34           0
               FOCUS              10:33:08       06:30           0
               SQL/DS              1:09:58       00:04           0


  PAGE     1                  2007/10/26 13.27.09      FLRPT204
```

Press Enter to scroll through subsequent pages of the report. At the last page of the report, press Enter to return to the menu screen.

Procedure: How to Build and Catalog a Custom FOCLOG Report

1. Write and debug a FOCEXEC that reports against the FOCLOG file. The author is responsible for code and results.

2. Store the FOCEXEC with a name of the following form:

   `FLRPT5`*`nn`*

   where:

   nn

   Is the next available two-digit number on the CUSTOM screen. For example, the ninth custom report should be named FLRPT509. The screen supports up to 10 reports.

3. Have your FOCLOG Administrator edit the FLMENU FOCEXEC to:

    a. Adjust the CRTFORM of the CUSTOM screen with the name of your report at the proper number.

    b. Then, after the following line, change the associated 9*xx* to 5*xx*:

    ```
    '-* REPORT NUMBER VALIDATION'
    ```

If you need any SET commands in your FOCEXEC, please reset them at the end of your routine. To do that, first capture the current setting with the following command:

```
? SET setname &HOLDIT
```

Then restore it at the end of the routine with:

```
SET setname=&HOLDIT
```

Report Layouts

The packaged reports all have standard four-line headings:

```
MAINFRAME FOCUS UTILIZATION ANALYSIS
COMPANY NAME
DATES COVERED (FROM - TO)
FOCLOG REPORT NAME (generated by your selection)
```

Report footings contain the page number, the date and time the report was requested and the internal name of the report corresponding to its number on the selection screen.

Report Contents

Each report examines an aspect of FOCUS usage at your site. There is considerable overlap among reports, as some present high-level usage summaries while others provide underlying details, potentially generating tens or even hundreds of pages.

Use the detail level reports to examine suspicious or irregular value clues on the summary reports.

Sort Options

Columns are sorted consistently across reports. Date periods are sorted in ascending chronological order, names are sorted alphabetically, and numeric columns are typically sorted high to low where the numbers of most interest tend to be the higher ones.

When optional sorts are invoked, the notation "* * SORT* *" appears at the top of the sort column indicating its selection by the user as opposed to a default.

Report Descriptions

Reference:

User Reports - 100 Series

File Dimension Reports - 200 Series

Procedure Dimension Reports - 300 Series

Usage Dimension Reports - 400 Series

Each of the following tables describes the reports available for a specific dimension. Each report has an internal name of the form:

```
FLRPTnnn
```

where:

nnn

Is the three-digit report number on the menu screen.

Reference: User Reports - 100 Series

The 100 Series reports show user activities and the resources consumed, permitting examination of usage by individuals, groups, usage types (online vs. batch) or a site-wide trend analysis.

No.	Report Title and Contents	Interpreting Report Output
101	**Duration of Online Sessions** summarizes the length of time users were logged on in predefined ranges of hours (<1, 1-4, 4-8, 8-12, >12). Only ranges with contents are displayed. ❑ The <1 hour group is usually ignored. ❑ 1-4 hours is effectively a morning or afternoon session, realizing that many users log off at lunch time, which creates two 1-4 hour sessions. ❑ 4-8 hours is a normal work day. ❑ 8-12 hours may show a dedicated worker. ❑ >12 hours may be a terminal left on overnight.	Look for the >12 hours category, which often indicates IDs left on overnight, exorbitant usage, or a process running on an unattended ID. Many other reports show the breakdown of Duration to further investigate these situations. For example, sort report 109 by Duration to see who was logged on for the various periods.
102	**Highest CPU-Consuming Users and Jobs** identifies users running procedures that absorbed the most CPU during the period. The report is sorted by CPU usage for Batch jobs and Online sessions	High CPU usage (particularly for online sessions) may indicate abuse or, more likely, users whose needs should be evaluated for possible efficiency improvement.

No.	Report Title and Contents	Interpreting Report Output
103	**Most Online Sessions** summarizes usage trends over the period by number of sessions (grouped by tens) and most frequent users in each.	Spikes identify heavy usage periods, such as end-of-month, quarter, or year, or special promotions or events. Upward trends by period may indicate other symptoms. Run Report 108 or 109 to see details of a specific user's activity in the higher categories.
104	**Total MSO Sessions** shows, by period, what part of FOCUS online usage is performed via MSO (for sites running MSO).	A significant percentage generally indicates a company's efficient use of shared resources provided by MSO.
105	**Total FOCUS Users** rolls up overall FOCUS usage across the site for each period. The Active Users and Total Jobs columns imply average usage per user, and the CPU column tracks the actual FOCUS processing performed during the period.	Look for spikes (positive or negative) and progressive trends in the numbers. Many other reports break down these summarized numbers.
106	**Attempt to Group Users by 4-Char Prefix** assumes that the first four characters of the userid roughly corresponds to definable groups in an organization. If that assumption is true for your company, this report may be of value. The report shows the number of users with that prefix and the CPU used by that group. Re-sorting by CPU could point to the largest group of FOCUS users on the site.	Assuming the first four user ID characters are significant workgroup differentiators, this report can be useful for comparing relative usage loads. Alternatively, see *Information Captured in the FOCLOG File* on page 369 for a description of the FOCLOG Master File in order to develop your own schemes for collapsing user IDs into local groups.
107	**Site-Wide FOCUS Usage Trend** summarizes site-wide FOCUS usage for each period, broken out by batch and online usage and by Reports (extracts) versus Updates (data changes)	Look for significant usage spikes in CPU or number of runs, and possible imbalances between reports and update operations

No.	Report Title and Contents	Interpreting Report Output
108	**Batch Job Details** describes every batch job during the period, showing the timestamp, who ran it, the LPAR FOCUS used, the job duration and CPU utilized.	This report could be hundreds or thousands of pages long. Sort by Duration or CPU to review largest jobs first.
109	**Online Session Details** describes every online session run during the period, showing its start time, who logged on, the LPAR FOCUS used, job duration and CPU utilized.	This report could be hundreds or thousands of pages long. Sort by Duration or CPU to see the most CPU-intensive sessions first. Similar to FLRPT108 for batch usage

Reference: File Dimension Reports - 200 Series

The 200 Series reports identify key files and show when and how they were used.

No.	Report Title and Contents	Interpreting Report Output
201	**Possible Extract Files Being Used** identifies the largest flat files being used. Flat files are non-keyed sequential files rather than structured databases. They are typically created by other applications or by extracts from local databases. Extract files are often CPU savers in that they require fewer resources than repeatedly retrieving data from the main file. Not all files on this report are 'extract' files; the report only knows that they are SUFFIX=FIX files, so some knowledge of the origin and use of the files are critical to the evaluation.	Low use of extract files should be investigated for potential removal or merging. Even high usage could point to the need to evaluate a better way to supply that data. Extract files often unknowingly grow excessively large over time, so a point of re-evaluation may be warranted.
202	**Files Originating Large HOLD File Extracts** shows databases from which large extracts are generated. Frequent creation may point to the need to evaluate the creation strategy.	Confirm acceptability of large HOLD files.

No.	Report Title and Contents	Interpreting Report Output
203	**Files Used by Users** shows the files accessed during the period by each user. Sort the report by Records, Lines, or CPU to see the most used files.	Use to investigate a user ID whose activity was highlighted on another report.
204	**Batch and Online File Usage** shows relative FOCUS online and batch usage by file type. This report concentrates on the type of file rather than the filename itself.	Usage at both ends of the spectrum may be of interest, depending on site conditions. Re-sort by Duration, CPU, or EXCP to review impact by category.
205	**Reports and Transactions Against Data Sources** details usage of specific data sources during each period. Sorted by period and file name.	Note the effect of file types on statistical columns. Scroll right to view Batch Updates column, which, along with the Batch Reports column, indicates the number of times the file was accessed during the period. Re-sort by Duration, CPU, or EXCP to review most used files by those categories.
206	**Files Using the Most CPU on Reports** rolls up file usage across the entire period. Sorted by file name, you may resort by CPU, Records, or Lines to observe the most used files. Use FLRPT205 to break down the file usage by smaller periods.	The Rollup shows the degree to which the data extracted was aggregated on the report versus printed row by row as raw data. This implies output volume and how that file is being used. The higher the Rollup, the better.
207	**Data Source Types Accessed by FOCUS** summarizes FOCUS use by file type.	Run Report FLRPT206 to assess usage by individual file.
208	**Most Records Extracted During Online Sessions** summarizes users who extracted very large volumes of data on a regular basis, grouping record extracts in 100,000 record increments. It shows number of occurrences per group.	High numbers of occurrences might warrant a review of usage or indicate the need for a better way to access the data regularly

Reference: Procedure Dimension Reports - 300 Series

No.	Report Title and Contents	Interpreting Report Output
301	**Most Frequently Run Procedures** lists the 30 most frequently executed procedures and their corresponding CPU usage sorted by frequency. You can re-sort by CPU to see the largest running procedures. Run FLRPT303 to list all procedures run during the period and review full usage details.	Frequently run procedures can generally be construed to be clearly valuable to the company. As such, you may want to get the most out of your investment by examining such procedures for efficiency, particularly if the CPU is high.
302	**FOCUS Command Usage** presents a broad summary of FOCUS usage across your site. Typically, TABLE, that is, data extracts into reports or extract files will be the most used command. MODIFY indicates loads of, or transactions against, data files. Other major FOCUS functions are shown if they are used.	Re-sort by Duration, CPU or EXCP to identify the most used commands by category.
303	**Procedure Run Details** lists every procedure run during the period showing the frequency, duration, CPU, and EXCP demands (and relative percent of each) within the period.	Re-sort by Duration, CPU, or EXCP to identify the most resource-intensive or widely-used procedures. Run FLRPT301 for a summary of the top 30 procedures.

No.	Report Title and Contents	Interpreting Report Output
304	**Candidate Procedures for Pooled Tables Adaptation** uncovers file usage patterns that appear to lend themselves to pooling, which could provide substantial CPU savings. This report has the potential of identifying procedures that can experience large CPU savings for relatively little investment.	Pooled Tables performs a single read of a data source for any number of consecutive reports against that source, delivering huge savings in I/O, and in CPU and elapsed times. Information Builders can help you evaluate the viability of pooling procedures listed on this report—contact your local Information Builders representative for details about Pooled Tables.

Reference: Usage Dimension Reports - 400 Series

No.	Report name and contents	What to look for
401	**All LPARS Running FOCUS** summarizes FOCUS usage/per period in each LPAR where FOCUS activity was recorded.	This is the only report of FOCUS activity by LPAR. You can re-sort the results by Duration or CPU.
402	**Output Formats Used by Reports** summarizes report destinations used during the period, revealing numbers of records extracted and lines output and the degree of aggregation (compression of application output). Many reports go directly to the user's screen or are sent to FOCUS HOLD or SAVE files (the General category), and some are sent to other output formats available from FOCUS, listed below that. Only those formats used are listed. Re-sort by Records, Lines, or CPU to see the most common or intensive destination for FOCUS output.	High aggregation ratios show data is being aggregated into more readily interpretable information. A low ratio for SCREEN implies that the raw data is displayed potentially hundreds of screens deep, which is invariably impenetrable and should be re-evaluated for usability. Non-FOCUS output formats imply FOCUS is supplying data for third-party databases and analytical tools such as Excel or PDF.

No.	Report name and contents	What to look for
403	**Daily User Activity Detail** shows daily activity for every online session for every user during the period. **Caution:** This report could easily run hundreds or thousands of pages.	This report provides the detail behind FLRPT102, Highest CPU-Consuming Users and Jobs. Look for repeated excessive expenses, which may in fact be perfectly valid high product usage. Re-sort by Records, Lines, Duration, or CPU to look for highest usage or minimal usage.
404	**Trend of Daily FOCUS Sessions (Report and Graph)** bar graph summarizes the number of batch sessions per day	Review usage trends and analyze spikes. Days without sessions, including weekends, are not shown.
405	**Hourly CPU Usage Accumulated (Graph)** bar graph accumulates CPU usage for each hour of the day, rolling together all days in the period. That is, a bar represents all of the usage accumulated on all days in the period for that particular hour. Hours are sorted from midnight to midnight; hours with no activity do not display. An hour shown represents the minutes in that hour, so '2am' represents 2am-3am.	Confirm the industry-typical trend of highest usage during workday hours and during peak times of overnight batch runs. Other spikes may be shift-related. Restrict the date range to one day to examine CPU usage on a particular day.

No.	Report name and contents	What to look for
406	**Possible Large Paper-Output Reports (100+ pages)** details procedures generating large offline print files. The report sorts the list by the largest reports in terms of approximate number of pages generated (assuming 40 lines of output per page, allowing for some typical lines of heading, footing, column heading, subfoots, blank lines, etc.). The report shows the procedure that ran the report and the (main) file against which the extract was done, as well as the number of times the report was so generated.	Confirm acceptable usage and number of runs. The largest printed reports are the ones to investigate.
407	**Daily Batch/Online CPU Utilization by Shift** categorizes CPU utilization by operating mode (online versus batch sessions) by approximate shifts (from 8AM to 6PM and from 6PM to 8AM) for each day in the report period.	A useful report for confirming performance is within normal bounds and if your CPU usage is inline with industry trends.
408	**Long-Running Sessions** is an elapsed-time summary that rolls up all batch jobs and online sessions during the period, displaying the longest and average durations and number of runs within two ranges of hours (>2 and >7).	Average numbers from this report should be useful in evaluating the data on other reports.

Index

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. You can contact us through the following methods:

Mail:	Documentation Services - Customer Support Information Builders, Inc. Two Penn Plaza New York, NY 10121-2898
Fax:	(212) 967-0460
E-mail:	books_info@ibi.com
Web form:	http://www.informationbuilders.com/bookstore/derf.html

Name:____

Company:____

Address:____

Telephone:______________________Date:______________________

E-mail:____

Comments:

Information Builders, Inc. Two Penn Plaza, New York, NY 10121-2898 (212) 736-4433

Reader Comments

Information Builders, Inc. Two Penn Plaza, New York, NY 10121-2898 (212) 736-4433

Summary of New Features DN1001076.0408
Version 7.6