

New Features

Introducing ...

**Two-Gigabyte FOCUS Database Support
and
Intelligent Partitioning**

**Information
Builders**

Contents

7.1 New Features

NF692: Aggregating and Sorting Report Columns

NF696: Calling Subroutines Written in REXX

Using REXX Subroutines

Compiling FUSREXX Macros in CMS

NF731: Reporting From Independent Paths

Retrieving Data From Multiple Paths

MULTIPATH and SET ALL Combinations

Determining if a Segment Is Required

NF749: HOLD FORMAT INTERNAL

NF750: DATASET in a Master File

DATASET Behavior in FOCUS Data Sources

DATASET Behavior in Fixed-Format Sequential Data Sources

DATASET Behavior in VSAM Data Sources

NF751: Date-Time Data Type

Describing Date-Time Values

Date-Time Display Formats

Specifying Date-Time Values

ACTUAL Formats for Date-Time Values

Setting Date-Time Options

Manipulating Date-Time Values

Comparison and Assignment

Date-Time Functions

NF755: Using FILEDEF for Creating Extract Files

- NF759: Increased Number of Display Fields**
- NF761: Comma Suppress Edit Format Option**
- NF762: Percent Edit Format Option**
- NF766: DEFINE Functions**
 - Using DEFINE Functions**
- NF773: Token Delimited Files**
- NF777: Two-Gigabyte FOCUS Database Support**
 - Enabling Two-Gigabyte Support**
- NF777: Partitioned FOCUS Data Sources**
 - Partitioning**
 - Intelligent Partitioning**
 - Specifying an Access File in a FOCUS Master File**
 - The FOCUS Access File**
 - FOCUS Access File Attributes**
 - Describing Joined Files**
- NF778: Dialogue Manager TRUNCATE Function**
 - Using the Dialogue Manager TRUNCATE Function**
- NF779: FOCUS CRTFORM HTML Translation**
- NF781: Embedding Text Fields in Headings**
- NF782: Oracle Data Adapter IXSPACE Setting**
 - Specifying Oracle Index Space Parameters**

NF785: The Adabas Write Data Adapter for FOCUS

Activating the Adabas Write Data Adapter

Limitations on Options Described for the Adabas Data Adapter

Fields That Cannot be Updated

Checking Adabas Return Codes and FOCUS Error Message Numbers

Adabas Write Examples

Types of Transaction Processing

Descriptor Considerations

Modifying Data

The MATCH Command

The NEXT Command

INCLUDE, UPDATE, and DELETE Processing

Adabas Transaction Control Within MODIFY

Using the Return Code Variable: FOCERROR

Using the Data Adapter SET ERRORRUN Command

Modifying Data sources Without Unique Keys

Referential Integrity

The MODIFY COMBINE Facility

The LOOKUP Function

The FIND Function

Data Adapter Error Messages and Adabas Response Codes

Adabas Write Data Adapter Error Messages

Adabas Response Codes

Common User Errors

7.0.9 New Features

NF575: Fusion

NF716: Euro Currency Support

Converting Currencies

Preparing FOCUS to Process Currency Conversions

Creating the Currency Database

Identifying Fields That Contain Currency Data

Activating the Currency Database

Processing Currency Data

NF744: HOLD FORMAT EXCEL

NF730: Hold Format PDF

Required Software Configuration

Downloading PDF Output

NF654: HOLD From External Sort

Conditions for Using External Sort to Create a HOLD File

NF597: Aggregation by External Sort

Conditions for Aggregating with an External Sort

NF728: Changing Retrieval Order with Aggregation

NF655: FOCPROF - The System Wide Profile

FOCUS Profiles

Using FOCUS Profiles

NF660: Multi-volume Support in MVS FOCUS

Advantages of Multi-volume Data Sources

Allocating Multi-volume Data Sources

Choosing Default Sizes for FOCUS-created Files

NF584: Dynamically Setting the IDMS DBNAME and DICTNAME

- NF673: Model 204 Interface Account Split**
- NF720: SQLJOIN OUTER Setting for Relational Interfaces**
- NF652: Teradata Interface Kanji Support**
- NF722: FOCUS Client DNS Names Support**
- NF656: Controlling REBUILD Messages**
- NF670: DYNAM Support for Unit Count**
 - Advantages of Multi-volume Data Sources**
- NF684: PCHOLD for Non-Html Files**
 - Using PCHOLD for Formats LOTUS, DIF, EXCEL, or PDF**
- NF683: Web Interface support for Maintain Winforms**
 - Prerequisites**
- NF691: Escape Character for LIKE Predicate**
 - Escape Character Capabilities**
- NF718: DYNAM Support for Existing Relative GDG Numbers**
 - Using DYNAM With Relative GDG Numbers**
- NF735: Enhancement to ? SET**
- NF740: Changes to the REBUILD Prompt**
- NF745: ? PTF Enhancements**
- NF746: Leading Zeros**
- NF748: HOLD FORMAT WP With Carriage Control**

7.0.8R New Features

NF557: REBUILD - Legacy Date Conversion

**How the REBUILD Utility Converts Legacy Dates
Updated Master File Created by REBUILD/DATE NEW
Action Taken on a Date Field During REBUILD/DATE NEW**

NF653: Displaying Base Dates in FOCUS Reports

NF659: CHECK FILE HOLD ALL

NF700: New Date Math Functions for the Year 2000

New Date Function Capabilities

Weekday Units

Business Day Units

Holidays

New Date Math Functions in MAINTAIN

NF703: Displaying Invalid Smart Dates in Reports

NF705: Enhancement to the YRTHRESH Command

NF708: Enhancement to the TODAY Subroutine

NF709: Displaying a Date Variable Without Separators

NF710: Field FORMAT=YYJUL

NF711: Altering Your System Date for Testing Purposes

NF713: MSO Log Changes

Sample MSO Log

NF714: LE Support

Recommended IBMLE Settings

Determining Proper IBMLE Settings

7.0.8 New Features

NF550: EDA/MSO Console Display for IMS PSB

NF564: Pooled Tables

Overview

Memory Needs

Report Size Estimates

FOCPOOLT

Reporting statistics

Sort Selection

Managing Memory

Common Selection Criteria

Reporting from non-Relational Databases

Reporting from Relational Databases

Pooled Tables in Batch Mode

Trace Facility

Tuning Applications

Pooled Tables Example

Single TABLE Clusters

Subpool Boundary Conditions

Pooled Tables Installation Instructions

Commands for the FOCPARM file

Frequently Asked Questions

NF566: MSO/CICS Cooperative Processing

MSO FOCEXEC Cooperative Processing Service

MSO/CICS Cooperative Processing Services

CMSORCV Function Codes

Reconnection Capability

Suspend key

Previous API

- NF568: DB2 Interface IF-THEN-ELSE Optimization**
- NF571: DB2 Interface SET ISOLATION Command**
- NF572: Invisible Ordered Character and Ordered Numeric Data Type Key Support**
- NF574: System 2000 Interface Trace Facility**
- NF579: Assigning Screening Conditions to a File for Reporting Purposes Using Filters
Filters and JOINS**
- NF583: Teradata Outer Join Optimization**
- NF586: Expanding Byte Precision for COUNT and LIST**
- NF593: IUCV CMS SU**
- NF594: JAVA Report Assist**
- NF605: Date Handling for the Year 2000 in FOCUS
Date Literals Interpretation Table**
- NF607: TABLA Enhancements**
- NF609: Sink Validation of Userids in CMS**
- NF617: Automatic Allocation of FOCUS Files**
- NF619: -HTMLFORM SAVE**
- NF620: Year 2000 Subroutines
Date Literals Interpretation Table**
- NF623: Increasing the Number of Verbs in a Report Request**
- NF626: JAVA Graph Wizard**

- NF628: Automatic Activation of Web Interface for Web Browser Users**
- NF630: Querying Which PTFs Have Been Applied for a Specific Release**
- NF631: Extended Plists**
- NF640: Dynamic Language Environment (LE) Support**
- NF642: Increased DEFINE Limitation**
- NF645: WEBHOME**
- NF647: Extended Support for Scandinavian External Sort**
- Project 2000 - Phase III**

Index

7.1 New Features

General Enhancements

[NF696: Calling Subroutines Written in REXX](#)

[NF750: DATASET in a Master File](#)

[NF751: Date-Time Data Type](#)

[NF773: Token Delimited Files](#)

[NF778: Dialogue Manager TRUNCATE Function](#)

[NF779: FOCUS CRTFORM HTML Translation](#)

Reporting Enhancements

[NF692: Aggregating and Sorting Report Columns](#)

[NF731: Reporting From Independent Paths](#)

[NF749: HOLD FORMAT INTERNAL](#)

[NF755: Using FILEDEF for Creating Extract Files](#)

[NF761: Comma Suppress Edit Format Option](#)

[NF762: Percent Edit Format Option](#)

[NF766: DEFINE Functions](#)

[NF781: Embedding Text Fields in Headings](#)

Raised Limits

[NF777: Two-Gigabyte FOCUS Database Support](#)

[NF759: Increased Number of Display Fields](#)

Performance Enhancements

[NF777: Partitioned FOCUS Data Sources](#)

Oracle Data Adapter

[NF782: Oracle Data Adapter IXSPACE Setting](#)

Adabas Data Adapter

[NF785: The Adabas Write Data Adapter for FOCUS](#)

NF692: Aggregating and Sorting Report Columns

Aggregation and sorting may be applied simultaneously to numeric columns in your report in one pass of the data.

Syntax How To Sort by a Report Column

The syntax is:

```
BY [HIGHEST|LOWEST {n} ] TOTAL display field
```

where:

n

Is the number of instances you wish to extract from the data source.

display field

Can be a fieldname, prefixoperator.fieldname or calculated value.

Example Sorting by a Report Column

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

```
TABLE FILE MOVIES
SUM LISTPR
BY CATEGORY
BY RATING
BY HIGHEST 5 TOTAL AVE.WHOLESALEPR AS 'AVE,WHOLESALEPR'
PRINT WHOLESALEPR
BY CATEGORY
BY RATING
WHERE CATEGORY EQ 'CLASSIC' OR 'MYSTERY'
END
```

The output is:

CATEGORY	RATING	WHOLESALEPR	LISTPR	WHOLESALEPR
CLASSIC	G	40.99	89.95	40.99
	NR	16.08	314.76	14.99
				20.00
				15.99
				10.95
				10.95
				9.99
				40.99
				10.95
				10.99
				15.00
MYSTERY	NR	9.00	19.98	9.00
	PG	9.00	39.96	9.00
				9.00
	PG13	9.00	19.98	9.00
	R	16.19	155.89	15.99

Reference Requirements

You must have an aggregating display command (SUM) for BY TOTAL to work correctly. A non-aggregating display command (PRINT) simply retrieves the data without aggregating it. The records will be sorted in either ascending or descending sequence based on your query.

NF696: Calling Subroutines Written in REXX

A FOCUS request can now call user-written subroutines coded in REXX. These routines, also called FUSREXX macros, add a 4GL option to the languages supported for user-written subroutines.

Using REXX Subroutines

REXX subroutines are supported in the VM/CMS and MVS environments:

- In CMS, a FUSREXX macro can contain either REXX source code or compiled REXX code created by running the source code through the REXX compiler. In addition, you can load either type of FUSREXX macro into memory using the EXECLOAD command. The compilation and load process reduces the CPU requirements and increases speed. Compilation also is a security tool, making private information difficult to read.
- In MVS, FOCUS supports source versions of REXX subroutines only.

Because of CPU requirements, the use of FUSREXX routines in large production jobs should be monitored carefully.

The following notes apply to the examples in this document:

- REXX versions are not necessarily the same in all operating environments. Therefore, some of the examples may use REXX functions that are not available in your environment.
- The REXX code is listed, but not fully explained. See your REXX documentation for information about REXX instructions and functions.

Syntax How to Call a REXX User-Written Subroutine

In a DEFINE FILE command:

```
DEFINE FILE filename  
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);  
END
```

In a DEFINE attribute in the Master File:

```
DEFINE fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen,  
outparm);
```

In a COMPUTE command:

```
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
```

In a Dialogue Manager -SET command:

```
-SET &var = subname(inlen1, inparm1, ..., outlen, outparm);
```

where:

fieldname

Is the name of the field to receive the return value.

An|In

Is the format of the field to receive return value.

subname

Is the name of the REXX routine.

inlen1, *inparm1* ...

Are the input parameters. Each parameter consists of a pair of values: a length and an alphanumeric parameter value. You can supply the name of an alphanumeric field, an alphanumeric literal, or an expression that resolves to an alphanumeric value. Up to 13 input parameter pairs are supported by FOCUS. Each parameter value can be up to 256 bytes long.

Note: Dialogue Manager converts input parameters that consist of numeric digits to decimal format, regardless of their original data type. Therefore, you cannot pass numeric input parameters to a REXX routine using -SET.

outlen, outparm

Is the output parameter pair, consisting of a length and a return value. In most cases, the return value should be alphanumeric, but integer return values are also supported. The return value can be the name of the field or Dialogue Manager variable to which the value is returned or its USAGE format enclosed in single quotation marks. The return value can be a minimum of one byte long and a maximum (for an alphanumeric value) of 256 bytes. **Note:** If the value returned is integer, *outlen* must be 4 because FOCUS reserves four bytes for integer fields.

&var

Is the name of the Dialogue Manager variable to receive the return value.

REXX subroutines:

- Require input data to be character and should return character output. Integer return values are also supported, but the output length in the subroutine call must be four. FOCUS has a 256-byte limit on character variables. This limit also applies to FUSREXX routines. FUSREXX routines return variable length data. For this reason, you must supply the length of the input arguments and the maximum length of the output data.
- Do *not* require any input parameters, but *do* require one return parameter, which *must* return at least one byte of data. It is possible for a FUSREXX function to need no input, such as a function that returns USERID.
- Do not support floating-point numbers (REXX does not have native floating-point conversion routines). All numeric fields should be converted to character format with no commas using a FOCUS function such as EDIT before being passed to the FUSREXX routine. This prevents FOCUS from converting numbers to floating point before passing them to the FUSREXX routine.
- Are not supported in Dialogue Manager -CMS RUN commands.

- On VM/CMS, the FILETYPE of REXX user-written functions is FUSREXX; they can be stored on any accessed disk.
- On MVS, DDNAME FUSREXX must be allocated to a PDS, and that library will be searched before other MVS libraries.
- The search order for subroutines is:
 1. FUSREXX
 2. Standard CMS or MVS search order.

Example Returning the Day of the Week

The FUSREXX routine DOW returns the day of the week an employee was hired. The routine passes one input parameter pair and one return field pair.

```
DEFINE FILE EMPLOYEE
1. AHDT/A6 = EDIT(HIRE_DATE) ;
2. DAY_OF_WEEK/A9 WITH AHDT= DOW(6,AHDT,9,DAY_OF_WEEK) ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME HIRE_DATE DAY_OF_WEEK
END
```

1. The input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The return field is up to nine bytes long and is named DAY_OF_WEEK.

The output is:

LAST_NAME	HIRE_DATE	DAY_OF_WEEK
-----	-----	-----
STEVENS	80/06/02	Monday
SMITH	81/07/01	Wednesday
JONES	82/05/01	Saturday
SMITH	82/01/04	Monday
BANNING	82/08/01	Sunday
IRVING	82/01/04	Monday
ROMANS	82/07/01	Thursday
MCCOY	81/07/01	Wednesday
BLACKWOOD	82/04/01	Thursday
MCKNIGHT	82/02/02	Tuesday
GREENSPAN	82/04/01	Thursday
CROSS	81/11/02	Monday

The FUSREXX macro is displayed below. The FUSREXX routine reads the input date, reformats it to MM/DD/YY format, and returns the day of the week using a REXX DATE call.

```
/* DOW routine. Return WEEKDAY from YMMDD format date */
Arg ymd .
Return Date('W',Translate('34/56/12',ymd,'123456'),'U')
```

Example Returning Text Format

The REXX function called in this request returns the number of copies of each classic movie in text format. It passes one input parameter and one return field.

```
TABLE FILE MOVIES
PRINT TITLE AND COMPUTE
1. ACOPIES/A3 = EDIT(COPIES); AS 'COPIES'
   AND COMPUTE
2. TXTCOPIES/A8 = NUMCNT(3,ACOPIES,8,TXTCOPIES);
   WHERE CATEGORY EQ 'CLASSIC'
END
```

1. The input field is 3 bytes long. Data is passed in field ACOPIES. The COPIES field is converted to an alphanumeric field.
2. The return field is up to 8 bytes long and is named TXTCOPIES.

The output is:

TITLE	COPIES	TXTCOPIES
-----	-----	-----
EAST OF EDEN	001	One
CITIZEN KANE	003	Three
CYRANO DE BERGERAC	001	One
MARTY	001	One
MALTESE FALCON, THE	002	Two
GONE WITH THE WIND	003	Three
ON THE WATERFRONT	002	Two
MUTINY ON THE BOUNTY	002	Two
PHILADELPHIA STORY, THE	002	Two
CAT ON A HOT TIN ROOF	002	Two
CASABLANCA	002	Two

The FUSREXX macro is:

```
/* NUMCNT routine. Pass a number from 0 to 10, return a character value */
Arg numbr .
data = 'Zero One Two Three Four Five Six Seven Eight Nine Ten'
numbr = numbr + 1          /* so 0 equals 1 element in array */
Return Word(data,numbr)
```

Example Passing Multiple Arguments

The following example shows how to pass multiple arguments to a FUSREXX routine. It is an interest calculation using the present salary for the employee and the employee start date to calculate a present value. It passes four input parameters and one return field.

```
DEFINE FILE EMPLOYEE
1. AHDT/A6      = EDIT(HIRE_DATE) ;
2. ACSAL/A12   = EDIT(CURR_SAL) ;
3. DCSAL/D12.2 = CURR_SAL ;
4. PV/A12      = INTEREST(6,AHDT,6,'&YMD',3,'6.5',12,ACSA,12,PV) ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE DCSAL PV
END
```

1. The first input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The current salary is converted to an alphanumeric field for use in the interest calculation.
3. The current salary is converted to a double-precision field to include commas and a decimal point in the output.
4. The second input field is six bytes long. Data is passed as a FOCUS character variable &YMD in YYMMDD format.

The third input field is a character value of 6.5, which is 3 bytes long to account for the decimal point in the character string.

The fourth input field is 12 bytes long. This passes the character field ACSAL.

The return field is up to 12 bytes long and is named PV.

The output is:

LAST_NAME	FIRST_NAME	HIRE_DATE	DCSAL	PV
-----	-----	-----	-----	---
STEVENS	ALFRED	80/06/02	11,000.00	14055.14
SMITH	MARY	81/07/01	13,200.00	15939.99
JONES	DIANE	82/05/01	18,480.00	21315.54
SMITH	RICHARD	82/01/04	9,500.00	11155.60
BANNING	JOHN	82/08/01	29,700.00	33770.53
IRVING	JOAN	82/01/04	26,862.00	31543.35
ROMANS	ANTHONY	82/07/01	21,120.00	24131.19
MCCOY	JOHN	81/07/01	18,480.00	22315.99
BLACKWOOD	ROSEMARIE	82/04/01	21,780.00	25238.25
MCKNIGHT	ROGER	82/02/02	16,100.00	18822.66
GREENSPAN	MARY	82/04/01	9,000.00	10429.03
CROSS	BARBARA	81/11/02	27,062.00	32081.82

The FUSREXX macro is displayed below. The REXX format command is used to format the return value.

```
/* Simple INTEREST program. dates are yymmdd format */
Arg start_date,now_date,percent,open_balance, .
begin = Date('B',Translate('34/56/12',start_date,'123456'),'U')
stop   = Date('B',Translate('34/56/12',now_date,'123456'),'U')
valnow = open_balance * (((stop - begin) * (percent / 100)) / 365)
Return Format(valnow,9,2)
```

Example Accepting Multiple Tokens in Parameters

FUSREXX routines can accept multiple tokens in a parameter. The following procedure passes employee information (pay date and monthly gross pay) as separate tokens in the first parameter. It passes three input parameters and one return field.

```
DEFINE FILE EMPLOYEE
1. COMPID/A256 = FN | ' ' | LN | ' ' | DPT | ' ' | EID ;
2. APD/A6 = EDIT(PAY_DATE) ;
3. APAY/A12 = EDIT(MO_PAY) ;
4. OK4RAISE/A1 = OK4RAISE(256,COMPID,6,APD,12,APAY,1,OK4RAISE) ;
END

TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME DEPARTMENT
IF OK4RAISE EQ '1'
END
```

1. The first input field is 256 bytes long. Data is passed in field COMPID. COMPID is the concatenation of several character fields passed as the first parameter. Each of the other parameters is a single argument.
2. The second input field is six bytes long. Data is passed in field APD. The pay date is converted to an alphanumeric field.
3. The third input field is 12 bytes long. Data is passed in field APAY. The monthly gross pay is converted to an alphanumeric field.
4. The return field is up to one byte long and is named OK4RAISE.

The output is:

EMP_ID	FIRST_NAME	LAST_NAME	DEPARTMENT
071382660	ALFRED	STEVENS	PRODUCTION

The FUSREXX macro is displayed below. Commas separate FUSREXX parameters. The ARG command specifies multiple variable names before the first comma and, therefore, separates the first FUSREXX parameter into separate REXX variables, using blanks as delimiters between the variables.

```
/* OK4RAISE routine. Parse separate tokens in 1st parm, then more parms*/
Arg fname lname dept empid, pay_date, gross_pay, .
If dept = 'PRODUCTION' & pay_date < '820000'
Then retvalue = '1'
Else retvalue = '0'
Return retvalue
```

FUSREXX routines *should* use the REXX RETURN function to return data to FOCUS. REXX EXIT is acceptable, but is generally used to end an EXEC, not a FUNCTION.

```
Correct
/* Some FUSREXX function */
Arg input
some rexx process ...
Return data_to_Focus
```

```
Not as Clear
/* Another FUSREXX function */
Arg input
some rexx process ...
Exit 0
```


Example Returning an Integer Value

It is possible for REXX to return a value that is *not* character format. The following example shows how REXX returns an integer value. This example also shows how the format of the integer field is used as the last field in the return argument. It passes two input fields and one return field. The FUSREXX routine NUMDAYS returns the number of days between hire date and date of increase. Note that the return value for an integer is *always* four bytes long.

```
DEFINE FILE EMPLOYEE
1. AHDT/A6 = EDIT(HIRE_DATE) ;
2. ADI/A6 = EDIT(DAT_INC) ;
3. BETWEEN/I6 = NUMDAYS(6,AHDT,6,ADI,4,'I6') ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME HIRE_DATE DAT_INC BETWEEN
IF BETWEEN NE 0
END
```

1. The first input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The second input field is six bytes long. Data is passed in field ADI. The date of increase is converted to an alphanumeric field.
3. The return field is up to six bytes long and is named BETWEEN.

The output is:

LAST_NAME	HIRE_DATE	DAT_INC	BETWEEN
-----	-----	-----	-----
STEVENS	80/06/02	82/01/01	578
STEVENS	80/06/02	81/01/01	213
SMITH	81/07/01	82/01/01	184
JONES	82/05/01	82/06/01	31
SMITH	82/01/04	82/05/14	130
IRVING	82/01/04	82/05/14	130
MCCOY	81/07/01	82/01/01	184
MCKNIGHT	82/02/02	82/05/14	101
GREENSPAN	82/04/01	82/06/11	71
CROSS	81/11/02	82/04/09	158

The FUSREXX macro is displayed below. The return value is converted from REXX character to HEX and formatted to be four bytes long.

```
/* NUMDAYS. Return number of days between 2 dates in yymmdd format */
/* The value returned will be in hex format
*/
Arg first,second .
base1 = Date('B',Translate('34/56/12',first,'123456'),'U')
base2 = Date('B',Translate('34/56/12',second,'123456'),'U')
Return D2C(base2 - base1,4)
```

Example Returning a Date Field From a FUSREXX Macro

FOCUS smart date fields contain the integer number of days since the base date 12/31/1900. REXX has a date function that can accept and return several types of date formats, including one called Base format ('B') that contains the number of days since the REXX base date 01/01/0001 (Jan. 1 of the Year 1).

Because input arguments must be alphanumeric, you cannot pass a smart date field to a REXX subroutine. Therefore, you can either:

- Pass the REXX routine an alphanumeric field with date display options and have it return a smart date value, if you account for the number of days difference between the FOCUS base date and the REXX base date and convert the result to integer.
- Pass the REXX routine a smart date value converted to alphanumeric format. With this technique, you must account for the difference in base dates for both the input and output.

The following example uses the technique of passing the subroutine an alphanumeric field with date display options. The FUSREXX macro called DATEREX1 takes two input arguments: an alphanumeric date in A8YYMD format and a number of days in character format. It returns a smart date in YYMD format that represents the input date plus the number of days. The FOCUS format A8YYMD corresponds to the REXX Standard format ('S').

The number 693959 represents the number of days difference between the FOCUS base date and the REXX base date:

```
/* REXX DATEREX1 routine. Add indate (format A8YYMD) to days */  
Arg indate, days .  
Return D2C(Date('B',indate,'S')+ days - 693959, 4)
```

The following request uses the DATEREX1 macro to calculate the date that is 365 days from the hire date of each employee. The input arguments are the hire date and the number of days to add. Because HIRE_DATE is in I6YMD format, it must be converted to A8YYMD before being passed to the macro:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE
AND COMPUTE
  ADATE/YYMD = HIRE_DATE; NOPRINT
AND COMPUTE
  INDATE/A8YYMD= ADATE; NOPRINT
AND COMPUTE
  NEXT_DATE/YYMD = DATEREX1(8,INDATE,3,'365',4,NEXT_DATE);
BY LAST_NAME NOPRINT
END
```

The output is:

LAST_NAME	FIRST_NAME	HIRE_DATE	NEXT_DATE
-----	-----	-----	-----
BANNING	JOHN	82/08/01	1983/08/01
BLACKWOOD	ROSEMARIE	82/04/01	1983/04/01
CROSS	BARBARA	81/11/02	1982/11/02
GREENSPAN	MARY	82/04/01	1983/04/01
IRVING	JOAN	82/01/04	1983/01/04
JONES	DIANE	82/05/01	1983/05/01
MCCOY	JOHN	81/07/01	1982/07/01
MCKNIGHT	ROGER	82/02/02	1983/02/02
ROMANS	ANTHONY	82/07/01	1983/07/01
SMITH	MARY	81/07/01	1982/07/01
SMITH	RICHARD	82/01/04	1983/01/04
STEVENS	ALFRED	80/06/02	1981/06/02

The following example uses the technique of passing the subroutine a smart date converted to alphanumeric format. The FUSREXX macro called DATEREX2 takes two input arguments: an alphanumeric number of days that represents a smart date, and a number of days to add. It returns a smart date in YYMD format that represents the input date plus the number of days. Both the input date and output date are in REXX base date ('B') format.

The number 693959 represents the number of days difference between the FOCUS base date and the REXX base date:

```
/* REXX DATEREX2 routine. Add indate (original format YYMD) to days */
Arg indate, days .
Return D2C(Date('B',indate+693959,'B') + days - 693959, 4)
```

The following request uses the DATEREX2 macro to calculate the date that is 365 days from the hire date of each employee. The input arguments are the hire date and the number of days to add. Because HIRE_DATE is in I6YMD format, it must be converted to an alphanumeric number of days before being passed to the macro:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE
AND COMPUTE
    ADATE/YYMD = HIRE_DATE; NOPRINT
AND COMPUTE
    INDATE/A8 = EDIT(ADATE); NOPRINT
AND COMPUTE
    NEXT_DATE/YYMD = DATEREX2(8, INDATE, 3, '365', 4, NEXT_DATE);
BY LAST_NAME NOPRINT
END
```

The report output is the same as that produced by the DATEREX1 macro.

Compiling FUSREXX Macros in CMS

The SUM2 FUSREXX macro takes two amounts as input and returns the sum in integer format:

```
/* SUM2 routine. Add amount1 to amount2 and return as integer */  
Arg amt1, amt2 .  
Return D2C(amt1 + amt2,4)
```

To compile and compress this FUSREXX macro in CMS, issue the following command. Note that the file identifier must be in upper case:

```
rexcomp SUM2 FUSREXX A (condense
```

A FILELIST of SUM2 * A lists the following files:

SUM2	CFUSREXX	A1 F	1024	2	1	1/31/00	12:07:19
SUM2	LISTING	A1 V	121	42	1	1/31/00	12:07:19
SUM2	FUSREXX	A1 F	80	3	1	1/31/00	12:04:19

The file SUM2 FUSREXX is the original source file. The file SUM2 CFUSREXX is the compiled version. To call the compiled version in a FOCUS request, you must rename it to have the file type FUSREXX. The file SUM2 LISTING details the results of the compilation.

To use the compiled version in a FOCUS request, issue the following commands. The EXECLOAD command, which loads the routine into memory and improves performance, is optional:

```
rename sum2 fusrex a ssum2 fusrex a  
rename sum2 cfusrex a sum2 fusrex a  
execload sum2 fusrex a
```

Then, in FOCUS, issue the following request:

```
TABLE FILE EMPLOYEE  
PRINT CSAL AND COMPUTE  
ASAL/A12 = EDIT(CSAL);  
AMOUNT/A4 = '1000';  
TOTSAL/I6 = SUM2(12, ASAL, 4, AMOUNT, 4, TOTSAL);  
END
```

The output is:

CURR_SAL	ASAL	AMOUNT	TOTSAL
-----	----	-----	-----
\$11,000.00	000000011000	1000	12000
\$13,200.00	000000013200	1000	14200
\$18,480.00	000000018480	1000	19480
\$9,500.00	000000009500	1000	10500
\$29,700.00	000000029700	1000	30700
\$26,862.00	000000026862	1000	27862
\$21,120.00	000000021120	1000	22120
\$18,480.00	000000018480	1000	19480
\$21,780.00	000000021780	1000	22780
\$16,100.00	000000016100	1000	17100
\$9,000.00	000000009000	1000	10000
\$27,062.00	000000027062	1000	28062

NF731: Reporting From Independent Paths

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

The MULTIPATH setting also affects the processing of selection tests on independent paths. In prior releases, WHERE or IF tests on separate paths were considered independently, as if an OR operator connected them. Therefore, a parent instance was included in the report if at least one of the paths passed its screening test. However the following warning message was produced in those cases:

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

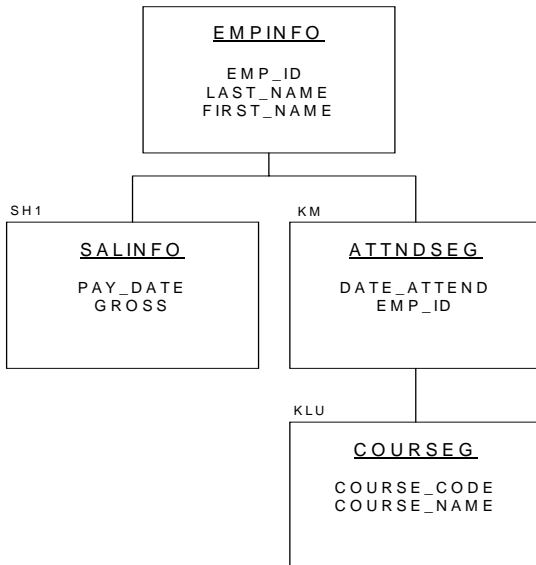
The request contains a test on one path, and retrieves data from another independent path. Records on the independent path will be retrieved regardless of whether the condition is satisfied on the tested path. Setting MULTIPATH = COMPOUND will alter this behavior.

This behavior is consistent with the SIMPLE setting for the MULTIPATH parameter.

The COMPOUND setting for MULTIPATH eliminates the (FOC144) message when testing on independent paths. It also treats screening conditions on separate paths as if they were connected by an AND operator. That is, *all* paths must pass their screening tests in order for the parent to be included in the report output.

Retrieving Data From Multiple Paths

The examples for this feature use the following segments from the EMPLOYEE data source:



Consider the following request that retrieves data from both paths with MULTIPATH = SIMPLE:

```
SET ALL = OFF
SET MULTIPATH = SIMPLE
TABLE FILE EMPLOYEE
PRINT GROSS IN 18 DATE_ATTEND COURSE_NAME
BY LAST_NAME BY FIRST_NAME IN 12
WHERE PAY_DATE EQ 820831
WHERE COURSE_CODE EQ '103'
END
```

The following warning message is generated:

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

John Banning has taken no courses but he is included in the report output because he has an instance on one of the two paths:

LAST_NAME	FIRST_NAME	GROSS	DATE_ATTEND	COURSE_NAME
BANNING	JOHN	\$2,475.00	.	.
BLACKWOOD	ROSEMA	\$1,815.00	.	.
CROSS	BARBAR	\$2,255.00	.	.
GREENSPAN	MARY	\$750.00	.	.
IRVING	JOAN	\$2,238.50	.	.
JONES	DIANE	\$1,540.00	82/05/26	BASIC REPORT PREP FOR PROG
MCCOY	JOHN	\$1,540.00	.	.
MCKNIGHT	ROGER	\$1,342.00	.	.
ROMANS	ANTHON	\$1,760.00	.	.
SMITH	MARY	\$1,100.00	81/11/16	BASIC REPORT PREP FOR PROG
	RICHAR	\$791.67	.	.
STEVENS	ALFRED	\$916.67	.	.

If you run the same request with MULTIPATH = COMPOUND, John Banning is omitted from the report output because he has no instances for COURSE_NAME:

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

The (FOC144) warning message is not generated.

Syntax How to Control Multi-path Processing

Issue the SET MULTIPATH command in one of the following ways:

- From the command level or in a stored procedure:

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

- In a report request:

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

where:

SIMPLE

Includes a parent segment in the report output if:

- It has at least one child that passes its screening conditions.
- It lacks any referenced child on a path, but the child is optional (see [Determining if a Segment Is Required](#)).

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

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

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

The request contains a test on one path, and retrieves data from another independent path. Records on the independent path will be retrieved regardless of whether the condition is satisfied on the tested path. Setting MULTIPATH = COMPOUND will alter this behavior.

COMPOUND

Includes a parent in the report output if it has *all* of its required children (see [Determining if a Segment Is Required](#)). The COMPOUND setting does not generate the (FOC144) warning message. COMPOUND is the default value for EDA and WebFOCUS.

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

MULTIPATH and SET ALL Combinations

The ALL parameter also affects independent path processing.

The following table uses examples from the EMPLOYEE data source to explain the interaction of ALL and MULTIPATH:

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

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

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

Determining if a Segment Is Required

FOCUS determines if a segment is required or optional using the following rules:

- When SET ALL is ON or OFF, a segment with WHERE or IF criteria is required for its parent, and all segments up to the root segment are required for their parents.

When SET ALL = PASS, a segment with WHERE or IF criteria is optional.

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

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

Reference Environments That Support MULTIPATH = COMPOUND

- The MULTIPATH setting works with all types of data sources and in all reporting environments (TABLE, TABLEF, AUTOTABLEF, MATCH, MORE, GRAPH, requests with multiple display commands). It works with alternate views, indexed views, filters, DBA, and joined structures.
- A unique segment is considered a part of its parent segment and, therefore, does not invoke independent path processing.

Reference Requirements and Notes for MULTIPATH = COMPOUND

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

(FOC36263) *Insufficient memory for independent path cache.*

There is not enough memory to build a cache for independent paths. Processing will proceed as if MULTIPATH were SIMPLE.

There is no limit to the *number* of segment instances (rows); however, no single segment instance can have more than 4K of active fields (referenced fields or fields needed for retrieving referenced fields). If this limit is exceeded, the SIMPLE setting is implemented and the following message is returned:

(FOC36264) *A segment's active fields are larger than the cache limit.*

One segment has more than 4000 bytes of active data.

Processing will proceed as if MULTIPATH were SIMPLE.

- SET MULTIPATH = COMPOUND creates a pool boundary when reports are pooled.
- WHERE criteria that screen on more than one path with the OR operator are not supported.

NF749: HOLD FORMAT INTERNAL

FOCUS HOLD files pad binary and packed data values to a full word boundary. For example, a three-digit integer field (I3), is stored as four bytes in a HOLD file. In order for third generation programs, such as COBOL, to be able to read FOCUS extract files in an exact manner, you can save the fields in the HOLD file without any padding.

Syntax How to Save a File Without Padding

Issue a report request specifying format overrides for the integer and packed fields that should not be padded and include the following:

```
ON TABLE HOLD AS name FORMAT INTERNAL
```

Syntax How to Ensure Accurate Display of Your Request

```
SET HOLDLIST = PRINTONLY
```

Setting HOLDLIST to PRINTONLY causes your report request to propagate the HOLD file with only the fields that would display in the report output as you specified. If you do not issue this setting, an extra field containing the padded field length is included in the HOLD file.

Reference How to Use Format Overrides

1. Integer fields (I) of one, two, three, or four bytes produce four byte integers.
2. HOLD FORMAT INTERNAL does not affect floating point double precision (D) and floating point single precision (F) fields. D remains at eight bytes and F at four bytes. Alpha fields are also not affected by HOLD FORMAT INTERNAL.

3. For packed decimal fields (Px.y), x is the total number of digits and y is the number of digits to the right of the decimal point. The number of bytes is derived by dividing x by 2 and adding 1.

`bytes = INT (x/2) + 1`

where:

`INT (x/2)` is the greatest integer after dividing by 2.

4. Alphanumeric fields automatically inherit their length from their source Master File and are not padded to a full word boundary.

Example Creating a HOLD File Without HOLD FORMAT INTERNAL

```
TABLE FILE CAR
PRINT CAR COUNTRY RETAIL_COST DEALER_COST SEATS
ON TABLE HOLD AS DJG
END
```

The request creates the following Master File:

```
FILE=DJG                ,SUFFIX=FIX
SEGNAME=DJG            ,SEGTYPE=S0
FIELDNAME   =CAR                ,E01            ,A16            ,A16            , $
FIELDNAME   =COUNTRY            ,E02            ,A10            ,A12            , $
FIELDNAME   =RETAIL_COST        ,E03            ,D7             ,D08            , $
FIELDNAME   =DEALER_COST        ,E04            ,D7             ,D08            , $
FIELDNAME   =SEATS              ,E05            ,I3             ,I04            , $
```

The values of ACTUAL for RETAIL_COST, DEALER_COST, and SEATS are all padded to a full word in binary.

Example Creating a HOLD File With HOLD FORMAT INTERNAL

In this example, DEALER_COST and RETAIL_COST are defined in the Master File as D fields, but the request overrides RETAIL_COST as an I2 field and DEALER_COST as a P3 field.

```
SET HOLDLIST=PRINTONLY
TABLE FILE CAR
PRINT CAR COUNTRY RETAIL_COST/I2 DEALER_COST/P3 SEATS/I1
ON TABLE HOLD AS HINT3 FORMAT INTERNAL
END
```

This creates the following Master File:

```
FILE=HINT3                , SUFFIX=FIX
SEGNAME=HINT3            , SEGTYPE=S0
FIELDNAME   =CAR          ,E01          ,A16          ,A16          , $
FIELDNAME   =COUNTRY      ,E02          ,A10          ,A10          , $
FIELDNAME   =RETAIL_COST  ,E03          ,I6           ,I02          , $
FIELDNAME   =DEALER_COST  ,E04          ,P4           ,P02          , $
FIELDNAME   =SEATS        ,E05          ,I4           ,I01          , $
```

The values of ACTUAL for the overridden fields are I2 and P2.

DEALER_COST has an ACTUAL of P2 because the format override, P3, means 3 display digits that can be stored in 2 actual digits.

If a format override is not large enough to contain the data values, the values are truncated. Truncation may cause the data in the HOLD file to be incorrect in the case of an integer. For packed data and integers, truncation occurs for the high order digits so the remaining low order digits will resemble the digits from the correct values. For example, consider this next request:

```
SET HOLDLIST=PRINTONLY
TABLE FILE CAR
PRINT CAR COUNTRY RETAIL_COST/I1 DEALER_COST/P3 SEATS/I1
ON TABLE HOLD AS HINT4 FORMAT INTERNAL
END
```

RETAIL_COST and SEATS are overridden with format I1 and DEALER_COST is overridden with format P3. These formats for RETAIL_COST and DEALER_COST are not large enough to contain the data values for these fields.

The following Master File is produced:

```
FILE=HINT4          ,SUFFIX=FIX
SEGNAME=HINT4      ,SEGTYPE=S0
FIELDNAME   =CAR           ,E01           ,A16           ,A16           , $
FIELDNAME   =COUNTRY      ,E02           ,A10           ,A10           , $
FIELDNAME   =RETAIL_COST  ,E03           ,I4            ,I01           , $
FIELDNAME   =DEALER_COST  ,E04           ,P4            ,P02           , $
FIELDNAME   =SEATS        ,E05           ,I4            ,I01           , $
```

The following is a sampling of the output when reporting from this HOLD file:

CAR	COUNTRY	RETAIL_COST	DEALER_COST	SEATS
---	-----	-----	-----	-----
JAGUAR	ENGLAND	174	427	2
JAGUAR	ENGLAND	179	194	5
JENSEN	ENGLAND	186	940	4
TRIUMPH	ENGLAND	236	296	2
DATSUN	JAPAN	67	626	4
TOYOTA	JAPAN	11	886	4
ALFA ROMEO	ITALY	164	660	2
ALFA ROMEO	ITALY	164	660	2
ALFA ROMEO	ITALY	37	915	4

The values displayed for RETAIL_COST and DEALER_COST do not represent the actual values in the original data source. This is due to truncation of the binary integer representation of the data values for RETAIL_COST, and truncation of the high order digits for DEALER_COST.

For example, a RETAIL_COST of 8878 is 10001010101110 in binary. To fit this into an I1 field as stated in the prior TABLE request, the value is 10101110. So the high order 100010 is truncated. Now, the decimal value of 100010 is 174, which is why the first record displayed shows a RETAIL_COST of 174.

For a DEALER_COST of 7427 to fit in a P2 ACTUAL, the high order digit, 7, is truncated, leaving the low order digits, 427. This is why the first record displayed shows a DEALER_COST of 427.

NF750: DATASET in a Master File

You can add the DATASET attribute to the Master File to specify a physical location for the data source to be allocated. In addition, the DATASET attribute permits you to bypass the FOCUS search mechanism for default data source location. DATASET eliminates the need to allocate data sources using JCL, FILEDEF, DYNAM, and USE commands.

User allocation and system specific behavior is as follows:

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

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

DATASET Behavior in FOCUS Data Sources

The DATASET attribute can be used only on the file level of the Master File. If the Master File's name is present in the USE list, or the user explicitly allocated the Master File, a warning is issued and the DATASET attribute is ignored.

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

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

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

An alternative to the DATASET attribute for allocating FOCUS data sources is an Access File. For detailed information, see [NF777: Partitioned FOCUS Data Sources](#). DATASET and ACCESSFILE are mutually exclusive attributes in the Master File; that is, you can use at most one of them, not both.

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

Syntax How to Use the DATASET Attribute

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

where:

filename

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

sinkname

Indicates that the data source is located on the FOCUS Database Server.

This attribute is valid for FOCUS data sources.

In MVS, the syntax is:

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

or

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

In CMS, the syntax is:

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

Example Allocating a FOCUS Data Source Using the DATASET Attribute

The following example illustrates how to allocate a FOCUS data source using the DATASET attribute:

For MVS,

```
FILENAME=CAR , SUFFIX=FOC
DATASET= ' USER1 . CAR . FOCUS '
SEGNAME=ORIGIN , SEGTYPE=S1
FIELDNAME=COUNTRY , COUNTRY , A10 , FIELDTYPE=I , $
SEGNAME=COMP , SEGTYPE=S1 , PARENT=ORIGIN
FIELDNAME=CAR , CARS , A16 , $
SEGNAME=CARREC , SEGTYPE=S1 , PARENT=COMP
.
.
.
```

For CMS,

```
FILENAME=CAR , SUFFIX=FOC
DATASET= ' CAR FOCUS A '
SEGNAME=ORIGIN , SEGTYPE=S1
FIELDNAME=COUNTRY , COUNTRY , A10 , FIELDTYPE=I , $
SEGNAME=COMP , SEGTYPE=S1 , PARENT=ORIGIN
FIELDNAME=CAR , CARS , A16 , $
SEGNAME=CARREC , SEGTYPE=S1 , PARENT=COMP
.
.
.
```


Example Allocating a Data Source For the FOCUS Database Server

The following example illustrates how to allocate a FOCUS data source with the DATASET attribute using ON *sink*:

For MVS,

```
FILENAME=CAR , SUFFIX=FOC
DATASET='CAR ON SINK1'
SEGNAME=ORIGIN , SEGTYPE=S1
FIELDNAME=COUNTRY , COUNTRY , A10 , FIELDTYPE=I , $
SEGNAME=COMP , SEGTYPE=S1 , PARENT=ORIGIN
FIELDNAME=CAR , CARS , A16 , $
SEGNAME=CARREC , SEGTYPE=S1 , PARENT=COMP
.
.
.
```

Note: The ddname CAR is allocated by the FOCUS Database Server JCL.

For CMS,

```
FILENAME=CAR , SUFFIX=FOC
DATASET='CAR FOCUS A ON SINK1'
SEGNAME=ORIGIN , SEGTYPE=S1
FIELDNAME=COUNTRY , COUNTRY , A10 , FIELDTYPE=I , $
SEGNAME=COMP , SEGTYPE=S1 , PARENT=ORIGIN
FIELDNAME=CAR , CARS , A16 , $
SEGNAME=CARREC , SEGTYPE=S1 , PARENT=COMP
.
.
.
```

DATASET Behavior in Fixed-Format Sequential Data Sources

The DATASET attribute must appear on the file level of the Master File and cannot contain ON *sink*. If the DATASET attribute contains ON *sink*, an error message is issued and the operation is terminated.

When FOCUS detects the DATASET attribute, FOCUS checks for an explicit allocation of data for this Master File. If an explicit allocation exists, a warning message is issued informing the user that the DATASET value has been overridden and the DATASET attribute is ignored. If this Master File name is not allocated, an internal command is issued to perform the allocation. This allocation is stored temporarily and is released when a new Master File is used or when the FOCUS session terminates.

Syntax How to Use the DATASET Attribute With Fixed-Format Data Sources

```
{DATASET|DATA}='filename'
```

where:

filename

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

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

- A user's explicit allocation overrides DATASET attributes.

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

Example Allocating a Fixed-Format Data Source Using the DATASET Attribute

The following example illustrates how to allocate a fixed-format data source using the DATASET attribute:

```
1. FILE=XX, SUFFIX=FIX, DATASET='SEQFILE1 DATA A'  
.  
.  
.  
2. FILE=XX, SUFFIX=FIX, DATASET='USER1.SEQFILE1'  
.  
.  
.
```

DATASET Behavior in VSAM Data Sources

The DATASET attribute must appear on the file level of the Master File and cannot contain ON *sink*. If the DATASET attribute contains ON *sink*, an error message is issued and the operation is terminated.

When FOCUS detects the DATASET attribute, FOCUS checks for an explicit allocation of data for this Master File. If an explicit allocation is found, a warning message is issued informing the user that the DATASET value has been overridden and the DATASET attribute is ignored. If this Master File name is not allocated, an internal command is issued to perform the allocation. This allocation is stored temporarily and is released when a new Master File is used or when the FOCUS session terminates.

The DATASET attribute may also appear on the field level of the Master File to specify where to find an alternate index. Because of VSAM naming conventions (truncated to 8 characters), the name of the field alias will be used as the ddname. If a user allocation is found for the Master File or alternate index ddname, the DATASET attribute is ignored and a warning message issued.

Note: There is no limit on how many alternate indices you may have. It is also acceptable for some alternate indices to have the DATASET attribute and others not. However, if a file level DATASET attribute is missing, the field level DATASET will be ignored.

Syntax **How to Use the DATASET Attribute With VSAM Data Sources**

```
{DATASET|DATA}='filename'
```

where:

filename

Is the platform-dependent physical name of the data source or alternate index.

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

- A user's explicit allocation overrides DATASET attributes.

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

Example Allocating a VSAM Data Source Using the DATASET Attribute

The following example illustrates how to allocate a VSAM data source on the file level and for an alternate index:

```
FILE=EXERVSM1, SUFFIX=VSAM, DATASET='VSAM1.CLUSTER1', $
SEGNAME=ROOT, SEGTYPE=S0, $
GROUP=KEY1, ALIAS=KEY, FORMAT=A4, ACTUAL=A4, $
FIELD=FLD1, ALIAS=F1, FORMAT=A4, ACTUAL=A4, $
FIELD=FLD2, ALIAS=F2, FORMAT=A4, ACTUAL=A4, $
FIELD=FLD3, ALIAS=DD1, FORMAT=A4, ACTUAL=A4, FIELDTYPE = I,
DATASET='VSAM1.INDEX1', $
FIELD=FLD4, ALIAS=F4, FORMAT=A4, ACTUAL=A4, $
FIELD=FLD5, ALIAS=F5, FORMAT=A4, ACTUAL=A4, $
FIELD=FLD6, ALIAS=F6, FORMAT=A4, ACTUAL=A4, $
FIELD=FLD7, ALIAS=F7, FORMAT=A4, ACTUAL=A4, $
```

Reference Error Messages

(FOC1920) ERROR ALLOCATING FILE %1 AS %2

System could not complete an internal allocation command.
Check file name, allocation, and attributes.

(FOC1921) ERROR PARSING DATASET IN MASTER %1: %2

Error occurred while analyzing value of DATASET. Check the
Master File for spelling errors.

(FOC1922) FOCUS DATASET INTERNAL ERROR: %1

A serious problem occurred processing the FOCUS DATASET.

(FOC1925) DATASET ERROR: PHYSICAL NAME SPECIFIED WITH 'ON' CLAUSE

Use a ddanme instead of the physical file name in DATASET
when ON *sinkname* clause is present.

Reference Warning Messages

(FOC1923) **WARNING: USE COMMAND OVERRIDES DATASET VALUE IN %1**
DATASET value in Master File has been overridden by USE
command.

(FOC1924) **WARNING: ALLOCATION OF %1 OVERRIDES DATASET VALUE IN %1**
DATASET value in Master File has been overridden by user's
allocation.

NF751: Date-Time Data Type

Topics:

- Describing Date-Time Values
- Setting Date-Time Options
- Manipulating Date-Time Values

The new date-time data type supports both the date and time, similar to the timestamp data types available in many relational data sources.

Date-time fields are stored in eight or ten bytes, four digits for date and either four or six digits for time, depending on whether the format specifies microseconds.

New subroutines are provided for manipulating date-time fields.

Describing Date-Time Values

In a Master File, the USAGE format for a date-time field describes which components to display and various options for displaying them. In Master Files for non-FOCUS data sources, date-time fields must also have an ACTUAL format that indicates how the date-time value is stored in the non-FOCUS data source.

The MISSING attribute for date-time fields can be ON or OFF. If it is OFF, and the date-time field has no value, it defaults to blank.

This section discusses:

- USAGE formats for displaying date-time field values.
- Alphanumeric formats for date-time values entered by a user at a terminal, read from a transaction file, or embedded in an expression.
- ACTUAL formats for date-time fields.

Date-Time Display Formats

The USAGE (or FORMAT) attribute determines how date-time field values are displayed in report output and forms, and how they behave in expressions and functions; for FOCUS data sources, it also determines how they are stored. A new format type, H, describes date-time fields. The USAGE attribute for a date-time field contains the H format code and can identify either the length of the field or the relevant date-time display options.

The USAGE attribute can be one of the following

USAGE = *Hnn*

USAGE = *Htimefmt1*

USAGE = *Hdatefmt [separator] [timefmt2]*

where:

Hnn

Is the USAGE value for a numeric date-time value without date-time display options. This format is appropriate for use in alphanumeric HOLD files or transaction files.

nm is the field length, from 1 to 20, including up to eight characters for displaying the date and up to nine or 12 characters for the time. For lengths less than 20, the date is truncated on the right.

An eight-character date includes four digits for the year, two digits for the month, and two digits for the day of the month, YYYYMMDD.

A nine-character time includes two digits for the hour, two digits for the minute, two digits for the second, and three digits for the millisecond, HHMMSSsss. The millisecond component represents the decimal portion of the second to three places.

A twelve-character time includes two digits for the hour, two digits for the minute, two digits for the second, three digits for the millisecond, and three digits for the microsecond, HHMMSSsssmmm. The millisecond component represents the decimal portion of the second value to three places. The microsecond component represents three additional decimal places beyond the millisecond value.

With this format, there are no spaces between the date and time components, no decimal points, and no spaces or separator characters within either component. The time must be entered using the 24-hour system. For example, the value `19991231225725333444` represents `1999/12/31 10:57:25.333444PM`

Htimefmt1

Is the USAGE format for displaying time only. Hour, minute, and second components are always displayed separated by colons (:), with no intervening blanks.

Unless you specify one of the AM/PM time display options, the time component is displayed using the 24-hour system.

When the format includes more than one time display option:

- The options must appear in the order hour, minute, second, millisecond, microsecond.
- The first option must be either hour, minute, or second.
- No intermediate component can be skipped. That is, if hour is specified the next option must be minute, it cannot be second.

The following table lists the valid time display options for a time-only USAGE attribute. Assume the time value is 2:05:27.123456 a.m.

Option	Meaning	Effect
H	hour (two digits) If the format includes the option <code>a</code> or <code>A</code> , the hour value is from 01 to 12. Otherwise, the hour value is from 00 to 23, with 00 representing midnight.	Prints a two-digit hour. For example: <code>USAGE = HH</code> prints <code>02</code>
h	hour with zero suppression If the format includes the option <code>a</code> or <code>A</code> , the hour value is from 1 to 12. Otherwise, the hour is from 0 to 23.	Displays the hour with zero suppression. For example: <code>USAGE = Hh</code> prints <code>2</code>
l	minute (two digits) The minute value is from 00 to 59.	Prints the two-digit minute. For example: <code>USAGE = HHl</code> prints <code>02:05</code>
i	minute with zero suppression The minute value is from 0 to 59.	Prints the minute with zero suppression. Cannot be used together with an hour format (H or h). For example: <code>USAGE = Hi</code> prints <code>5</code>

Option	Meaning	Effect
S	Second (two digits) 00 to 59	Prints the two-digit second. For example: <code>USAGE = HHIS prints 02:05:27</code>
s	millisecond (three digits — after the decimal point in the second) 000 to 999	Prints the second to three decimal places. For example: <code>USAGE = HHISs prints 02:05:27.123</code>
m	microsecond (three additional digits after milliseconds) 000 through 999	Prints the second to six decimal places. For example: <code>USAGE = HSsm prints 27.123456</code>
A	12-hour time display with AM or PM in upper case	Prints hours from 01 to 12 followed by AM or PM. For example: <code>USAGE = HHISA prints 02:05:27AM</code>
a	12-hour time display with am or pm in lower case	Prints hours from 01 to 12 followed by am or pm. For example: <code>USAGE = HHISa prints 02:05:27am</code>

Hdatefmt

Is the USAGE format for displaying the date portion of the date-time field.

The date components can be in any of the following combinations and order:

- Year first combinations: Y, YY, YM, YYM, YMD, YYMD
- Month-first combinations: M, MD, MY, MYY, MDY, MDYY
- Day-first combinations: D, DM, DMY, DMYY

The date format can include the following display options as long as they conform to the allowed combinations. In the following table, assume the date is February 5, 1999.

Option	Meaning	Example
Y	2-digit year	99
YY	4-digit year	1999
M	2-digit month (01 - 12)	02
MT	Full month name	February
Mt	Short month name	Feb
D	2-digit day	05
d	zero-suppressed day	5
k	For formats in which month or day is followed by year and month is translated to a short or full name, separates the year from the day with a comma and blank. Otherwise the separator is a blank.	USAGE = HMtDkYY prints Feb 05, 1999

separator

Is a separator between the date components. The default separator is a slash (/). Other valid separators are: period (.), hyphen (-), blank (B), or none (N). With translated months, these separators can only be specified when the k option is not used.

timefmt2

Is the format for a time that follows a date. Time is separated from the date by a blank; time components are separated from each other by colons. Unlike the format for time alone, a time format that follows a date format consists of at most two characters: a single character to represent all of the time components to be displayed and, optionally, one character for an AM/PM option. The following table lists the valid options. Assume the date is February 5, 1999 and the time is 02:05:25.444555 a.m.

Option	Meaning	Example
H	Prints hour	USAGE = HYYMDH prints 1999/02/05 02
I	Prints hour:minute	USAGE = HYYMDI prints 1999/02/05 02:05
S	Prints hour:minute:second	USAGE = HYYMDS prints 1999/02/05 02:05:25
s	Prints hour:minute:second.millisecond	USAGE = HYYMDs prints 1999/02/05 02:05:25.444

Option	Meaning	Example
m	Prints hours:minutes:seconds.microseconds	<code>USAGE = HYYMDm prints</code> <code>1999/02/05 02:05:25.444555</code>
A	Prints AM or PM. Uses the 12-hour system and causes the hour to be printed with zero suppression.	<code>USAGE = HYYMDSA prints</code> <code>1999/02/05 2:05:25AM</code>
a	Prints am or pm. Uses the 12-hour system and causes the hour to be printed with zero suppression.	<code>USAGE = HYYMDSa prints</code> <code>1999/02/05 2:05:25am</code>

Note: Unless you specify one of the AM/PM time display options, the time component is displayed using the 24-hour system.

Specifying Date-Time Values

An external date-time value is a constant in character format from one of the following sources:

- A sequential data source.
- Typed by an application user at a terminal or workstation.
- Used in an expression in a WHERE clause, an IF clause, a DEFINE, or a COMPUTE.

A date-time constant typed by an application user at a terminal or workstation, or a date-time value as it appears in a character file has one of the following formats

```
date_string [time_string]
time_string [date_string]
```

A date-time constant in a COMPUTE, DEFINE, or WHERE expression must have one of the following formats.

```
DT(date_string [time_string])
DT(time_string [date_string])
```

A date-time constant in an IF expression has one of the following formats:

```
'date_string [time_string]'
'time_string [date_string]'
```

If the value contains no blanks or special characters, the single quotation marks are not necessary. Note that the DT prefix is not supported in IF criteria.

where:

time_string

Cannot contain blanks. Time components are separated by colons and may be followed by AM, PM, am, or pm. For example:

```
14:30:20:99          (99 milliseconds)
14:30
14:30:20.99          (99/100 seconds)
14:30:20.999999      (999999 microseconds)
02:30:20:500pm
```

Note that seconds can be expressed with a decimal point or be followed by a colon.

- If there is a colon after seconds, the value following it represents milliseconds. There is no way to express microseconds using this notation.
- A decimal point in the seconds value indicates the decimal fraction of a second. Microseconds can be represented using six decimal digits.

date_string

Can have one of the following three formats:

- The **numeric string format** is exactly four, six, or eight digits. Four-digit strings are considered to be a year (century must be specified); the month and day are set to January 1. Six and eight-digit strings contain two or four digits for the year, followed by two for the month, and then two for the day. Because the component order is fixed with this format, the DATEFORMAT setting described in [How to Specify the Order of Date Components in Formatted Input Values](#) is ignored.

If a numeric-string format longer than eight digits is encountered, it is treated as a combined date-time string in the *Hnn* format described in [Describing Date-Time Values](#). The following are examples of numeric string date constants:

```
99
1999
19990201
```

- The **formatted-string format** contains a one or two-digit day, a one or two-digit month, and a two or four-digit year separated by spaces, slashes, hyphens, or periods. All three parts must be present and follow the DATEFORMAT setting described in [How to Specify the Order of Date Components in Formatted Input Values](#). If any of the three fields is four digits, it is interpreted as the year, and the other two fields must follow the order given by the DATEFORMAT setting. The following are examples of formatted-string date constants:

```
1999/05/20
5 20 1999
99.05.20
1999-05-20
```


- The **translated-string format** contains the full or abbreviated month name. The year must also be present in four-digit or two-digit form. If the day is missing, day 1 of the month is assumed; if present, it can have one or two digits. If the string contains both a two-digit year and a two-digit day, they must be in the order given by the DATEFORMAT setting. For example:

January 6 2000

Note:

- The date and time strings must be separated by at least one blank space. Blank spaces are also permitted at the beginning and end of the date-time string.
- In each date format, two-digit years are interpreted using the [F]DEFCENT and [F]YRTHRESH settings.

Example Reading Date-Time Values From a Transaction File

The DTTRANS comma-delimited transaction file has an ID field and a date-time field that contains both the date (as eight characters) and time (in the format hour:minute:second):

```
01, 20000101 02:57:25,$  
02, 19991231 14:05:35,$
```

Because the transaction file contains the dates in numeric string format the DATEFORMAT setting is not used, and the dates are entered in YMD order.

The following transaction file is also valid. It contains formatted string dates that comply with the default DATEFORMAT setting, MDY:

```
01, 01/01/2000 02:57:25,$  
02, 12/31/1999 14:05:35,$
```

The following Master File describes the FOCUS data source named DATETIME, which will receive these values:

```
FILE=DATETIME,      SUFFIX=FOC      , $
SEGNAME=DATETIME,  SEGTYPE=S0      , $
FIELD=ID, ID,      USAGE = I2      , $
FIELD=DT1, DT1,    USAGE=HYMDS     , $
```

The following MODIFY procedure loads the transaction values into the FOCUS data source:

```
-* THE FOLLOWING ALLOCATION AND CREATE FILE IS NEEDED FOR THE TRANSACTION
FILE
-* ON MVS
-* DYNAM ALLOC DD DTTRANS DA USER1.DTTRANS SHR REUSE
-* CREATE FILE DATETIME
-* THE FOLLOWING FILEDEF IS NEEDED FOR THE TRANSACTION FILE ON CMS
CMS FILEDEF DTTRANS DISK DTTRANS DATA A
MODIFY FILE DATETIME
      FREEFORM ID DT1
DATA ON DTTRANS
END
```

To see the printed values, issue the following request:

```
TABLE FILE DATETIME
PRINT ID DT1
END
```

The output is:

```
ID  DT1
--  ---
 1  2000/01/01 02:57:25
 2  1999/12/31 14:05:35
```

Example Using a Date-Time Value in a COMPUTE Command

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME AND COMPUTE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYMDIA = DT(20000101 09:00AM);
WHERE CURR_JOBCODE LIKE 'B%'
END
```

The output is:

LAST_NAME	FIRST_NAME	NEWSAL	RAISETIME
-----	-----	-----	-----
SMITH	MARY	\$14,520.00	2000/01/01 9:00AM
JONES	DIANE	\$20,328.00	2000/01/01 9:00AM
ROMANS	ANTHONY	\$23,232.00	2000/01/01 9:00AM
MCCOY	JOHN	\$20,328.00	2000/01/01 9:00AM
BLACKWOOD	ROSEMARIE	\$23,958.00	2000/01/01 9:00AM
MCKNIGHT	ROGER	\$17,710.00	2000/01/01 9:00AM

Example Using a Date-Time Value in WHERE Criteria

In a WHERE clause, a date-time constant must use the DT() format:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE
WHERE TRANSDATE GT DT(2000/01/01 02:57:25)
END
```

The output is:

CUSTID	TRANSDATE
-----	-----
1118	2000/06/26 05:45
1237	2000/02/05 03:30

Example Using a Date-Time Value in IF Criteria

In an IF clause, a date-time constant must be enclosed in single quotation marks if it contains any blanks:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE
IF TRANSDATE GT '2000/01/01 02:57:25'
END
```

Note: The DT prefix for a date-time constant is not supported in an IF clause.

The output is:

```
CUSTID  TRANSDATE
-----  -
1118    2000/06/26 05:45
1237    2000/02/05 03:30
```

ACTUAL Formats for Date-Time Values

ACTUAL formats supported for date-time values are:

- *Ann*, H8, H10, and H12. *Ann* accepts all the date-time string formats described in [How to Specify the Order of Date Components in Formatted Input Values](#), as well as the *Hnn* USAGE display format described in [Describing Date-Time Values](#). ACTUAL=H8, H10, or H12 accepts a date-time field as it occurs in a binary HOLD file or SAVB file. ACTUAL=*Ann* accepts a date-time field as it occurs in an alphanumeric HOLD file or SAVE file.

Example Creating a Binary HOLD File Containing a Date-Time Field

The following request creates a binary HOLD file using the VIDEOTR2 data source:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE
ON TABLE HOLD AS DTHOLD
END
```

The DTHOLD Master File created from this request contains the ACTUAL format H8 for the TRANSDATE date-time field:

```
FILE=DTHOLD           , SUFFIX=FIX
SEGNAME=DTHOLD       , SEGTYPE=S0
FIELDNAME   =CUSTID           ,E01           ,A4           ,A04           , $
FIELDNAME   =TRANSDATE        ,E02           ,HYMDI        ,H08           , $
```

Example Creating an Alphanumeric HOLD File Containing a Date-Time Field

The following request creates an alphanumeric HOLD file using the VIDEOTR2 data source:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE
ON TABLE HOLD AS DTALPHA FORMAT ALPHA
END
```

The DTALPHA Master File created from this request contains the ACTUAL format A17 for the TRANSDATE date-time field. The ACTUAL format is A20 if the time format includes microseconds, A17 otherwise (padded with low order zeros if necessary):

```
FILE=DTALPHA           ,SUFFIX=FIX
SEGNAME=DTALPHA ,SEGTYPE=S0
FIELDNAME   =CUSTID           ,E01           ,A4           ,A04           , $
FIELDNAME   =TRANSDATE        ,E02           ,HYMMDI        ,A17           , $
```

Setting Date-Time Options

Three parameter settings determine how to interpret a date-time value and manipulate it in requests:

- DATEFORMAT specifies the order of the date components.
- WEEKFIRST assigns a day to be considered the first day of the week in date-time computations involving weeks.
- DTSTRICT controls whether values are error checked (for example, whether the day portion of a date is within the correct number of days for the specified month).

Syntax How to Specify the Order of Date Components in Formatted Input Values

The DATEFORMAT parameter specifies the order of the date components (month/day/year) when date-time values are entered in the formatted string and translated string formats described in [Describing Date-Time Values](#). It makes a value's input format independent of the format of the variable to which it is being assigned. The syntax is

```
SET DATEFORMAT = datefmt
```

where:

datefmt

Can be one of the following: MDY, DMY, YMD, or MYD. The U. S. English default format is MDY.

Syntax How to Specify the First Day of the Week

The WEEKFIRST parameter is used in week computations by the HDIFF, HNAME, HPART, and HSETPT functions described in [Date-Time Functions](#). For an example, see [Extracting the Week Component With Different WEEKFIRST Settings](#). The values from 1 to 7 represent Sunday through Saturday. The syntax is

```
SET WEEKFIRST = number
```

where:

number

Is a number from one to seven, where one represents Sunday and seven represents Saturday. The U. S. English default value is seven (Saturday) meaning that Saturday is the first day of each week, so every Friday-Saturday transition is the start of a new week.

The WEEKFIRST setting does not change the number that corresponds to each day of the week, it just specifies which one is considered the start of the week. The default of Saturday (7) as the first day of the week is consistent with the Microsoft SQL Server convention.

Syntax How to Control Error Checking of Date-Time Values

The DTSTRICT parameter controls how much error checking is done on date-time values when they are input by users, read from an alphanumeric transaction file, displayed, or used in user-written subroutines. The syntax is

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

where:

ON

Invokes strict processing. This means that whenever a date-time value is input by a user, read from a transaction file, displayed, or returned by a subroutine it is checked to make sure that the value represents a valid date and time. For example, a numeric month must be between 1 and 12, and the day must be within the number of days for the specified month. ON is the default value. If you attempt to enter a value that violates this rule, the following message displays:

```
(FOC177) INVALID DATE CONSTANT: dt_constant
```

OFF

Does not invoke strict processing. Any date-time component can have any value within the constraint of the number of decimal digits allowed; for example, the month value can be 00 or 13 or 99, but not 115. Furthermore, the values do not have to be consistent; for example, any month in any year can have 30 or 31 days.

Manipulating Date-Time Values

The only direct operations that can be performed on date-time variables and constants are comparison using a logical expression and simple assignment of the form $A = B$. All other operations are accomplished through a set of date-time functions.

Comparison and Assignment

Any two date-time values can be compared, even if their lengths do not match. If a date-time field supports missing values, fields that contain the missing value have a greater value than any date-time field can have.

Therefore, in order to exclude missing values from the report output when using a GT or GE operator in a selection test, it is recommended that you add the additional constraint *field* NE MISSING to the selection test:

```
date_time_field {GT|GE} date_time_value AND date_time_field NE MISSING
```

Assignments are permitted between date-time formats of equal or different lengths. Assigning a 10-byte date-time value to an 8-byte date-time value truncates the microsecond portion (no rounding takes place). Assigning a short value to a long one sets the low-order three digits of the microseconds to zero.

Other operations, including arithmetic, concatenation, EDIT, and LIKE on date-time operands are not supported. Prefix operators that work with alphanumeric fields are supported.

Example Testing for Missing Date-Time Values

Consider the DATETIM2 Master File:

```
FILE=DATETIM2, SUFFIX=FOC , $
SEGNAME=DATETIME, SEGTYPE=S0 , $
FIELD=ID, ID, USAGE = I2 , $
FIELD=DT1, DT1, USAGE=HYMDS, MISSING=ON, $
```

Field DT1 supports missing values. Consider the following request:

```
TABLE FILE DATETIM2
PRINT ID DT1
END
```

The resulting report output shows that in the instance with ID=3, the field DT1 has a missing value:

```
ID  DT1
--  ---
 1  2000/01/01 02:57:25
 2  1999/12/31 00:00:00
 3  .
```

The following request selects values of DT1 that are greater than 2000/01/01 00:00:00 and are not missing:

```
TABLE FILE DATETIM2
PRINT ID DT1
  WHERE DT1 NE MISSING AND DT1 GT DT(2000/01/01 00:00:00);
END
```

The missing value is not included in the report output:

```
ID  DT1
--  ---
 1  2000/01/01 02:57:25
```

Example Assigning a Different Usage Format to a Date-Time Column

Consider the following request using the VIDEOTR2 data source:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AND COMPUTE
  DT2/HYYMDH = TRANSDATE;
  T1/HHIS = TRANSDATE;
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	TRANSDATE	DT2	T1
-----	-----	---	--
1118	2000/06/26 05:45	2000/06/26 05	05:45:00
1237	2000/02/05 03:30	2000/02/05 03	03:30:00

Date-Time Functions

The following functions allow you to manipulate date-time values:

Function Name	Description
HCNVRT	Converts date-time values to alphanumeric format for use with operators such as EDIT, CONTAINS, and LIKE.
HINPUT	Converts an alphanumeric string to a date-time value.
HADD	Increments date-time values by a specified number of units.
HDIFF	Returns the number of units of a specific date-time component between two date-time values.
HNAME	Extracts specified components of a date-time value and converts them to alphanumeric format.
HPART	Extracts a component of a date-time value in numeric format.
HSETPT	Inserts the numeric value of a specified component in a date-time field.
HMIDNT	Changes the time portion of a date-time field to midnight.
HDATE	Extracts the date components from a date-time field and converts them to a date field.
HDTTM	Converts a date field to a date-time field with the time set to midnight.

Function Name	Description
HTIME	Extracts all of the time components from a date-time field and converts them to a number of milliseconds or microseconds in numeric format.
HGETC	Returns the current date and time in date-time format.

Note:

- In those arguments that give you a choice of 8 or 10, use 8 for processing values without microseconds, 10 when the field value includes microseconds.
- The last argument is always a USAGE format that indicates the data type returned by the function. The type may be A (alpha), I (integer), D (double precision), DATE (smart date), or H (date-time).

Reference Component Names and Values for Use With Date-Time Functions

The following component names and values are supported as arguments to those date-time functions that require you to specify a component name as an argument:

Component Name	Valid Values
<code>year</code>	0001-9999
<code>quarter</code>	1-4
<code>month</code>	1-12
<code>day-of-year</code>	1-366
<code>day</code> or <code>day-of-month</code>	1-31 (The two names for the component are equivalent.)
<code>week</code>	1-53
<code>weekday</code>	1-7 (Sunday-Saturday)
<code>hour</code>	0-23
<code>minute</code>	0-59
<code>second</code>	0-59
<code>millisecond</code>	0-999
<code>microsecond</code>	0-999999

Reference Notes Regarding ISO Standard Date-Time Representations

International Standard ISO 8601 describes the standards for numeric representations of date and time. Some of the relevant standards and notes about their implementation follow:

- The international standard date notation is YYYY-MM-DD. In this implementation, you can control the date format used to enter date-time values with the DATEFORMAT parameter. For details, see [How to Specify the Order of Date Components in Formatted Input Values](#).
- The international standard for the first day of a week is Monday. You can use the WEEKFIRST parameter to control the day used as the first day of the week by the date-time functions.
- The standard specifies that week 1 of a year is the first week of the year that has a Thursday. Combined with the standard of Monday as day 1, this rule ensures that week 1 has at least four of its days in the specified year.

The following rules represent an extension to the standard in this implementation:

- Whatever day you choose for your WEEKFIRST setting, the date-time functions define week 1 as the first week with at least four days in the specified year.
- With these rules, it is possible for the first few days of January to fall in the week prior to week 1. The international standard considers these dates to be in week 53 of the previous year. However, the date-time functions return zero for the week component when it falls in the week prior to week 1.
- The international standard notation for the time of day is hh:mm:ss using the 24-hour system. However, the date-time data type and date-time functions allow you to use the 12-hour system.

HCNVRT: Converting a Date-Time Field to Alphanumeric Format

Use the following syntax in an expression to convert a date-time field to alphanumeric format for use with operators such as EDIT, CONTAINS, and LIKE.

```
HCNVRT (dtfield, '(Hfmt)', rlength, 'Ann')
```

where:

dtfield

Is the date-time value to convert. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

Hfmt

Is the USAGE format of the date-time field being converted, enclosed in parentheses and single quotation marks.

rlength

Is the length of the alphanumeric field returned. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value. If *rlength* is smaller than the number of characters needed to display the alphanumeric field, an all-blank field is returned.

Ann

Is the USAGE format of the returned alphanumeric value, enclosed in single quotation marks.

Example Converting a Date-Time Field to Alphanumeric Format

The following request converts the TRANSDATE field to alphanumeric format:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME1/A20 = HCNVRT (TRANSDATE,'(H17)', 17, 'A20');
ALPHA_DATE_TIME2/A20 = HCNVRT (TRANSDATE,'(HYMDS)', 20, 'A20');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ALPHA_DATE_TIME1	ALPHA_DATE_TIME2
1118	2000/06/26 05:45	20000626054500000	2000/06/26 05:45:00
1237	2000/02/05 03:30	20000205033000000	2000/02/05 03:30:00

HINPUT: Converting an Alphanumeric String to a Date-Time Value

Use the following syntax in an expression to convert an alphanumeric string to a date-time value.

```
HINPUT (inputlength, 'inputstring', {8|10}, 'Hfmt')
```

where:

inputlength

Is the length of the alphanumeric string to convert. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

inputstring

Is the alphanumeric string to convert. You can supply the actual string enclosed in single quotation marks, the name of an alphanumeric field, or an expression that returns an alphanumeric value. The alphanumeric string can consist of any valid date-time input value as described in [Describing Date-Time Values](#).

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

Hfmt

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Converting an Alphanumeric String to a Date-Time Value

The following request converts the TRANSDATE field to alphanumeric format (using the HCNVRT function) and then uses the HINPUT routine to convert the alphanumeric string to a date-time value:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME/A20 = HCNVRT (TRANSDATE,'(H17)', 17, 'A20');
DT_FROM_ALPHA/HYYMDS = HINPUT(14, ALPHA_DATE_TIME, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ALPHA_DATE_TIME1	ALPHA_DATE_TIME2
1118	2000/06/26 05:45	20000626054500000	2000/06/26 05:45:00
1237	2000/02/05 03:30	20000205033000000	2000/02/05 03:30:00

HADD: Incrementing a Date-Time Field

Use the following syntax in an expression to increment a date-time field by a given number of units, for example, 1 year, 3 months, or -15 seconds.

```
HADD (dtfield, 'component', increment, {8 | 10}, 'Hformat')
```

where:

dtfield

Is the date-time value to increment. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be incremented, enclosed in single quotation marks. See [Component Names and Values for Use With Date-Time Functions](#) for a list of supported components.

increment

Is the number of units by which to increment the specified component. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

Hformat

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Incrementing the Month Component of a Date-Time Field

The following request adds two months to the TRANSDATE field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ADD_MONTH
1118	2000/06/26 05:45	2000/08/26 05:45:00
1237	2000/02/05 03:30	2000/04/05 03:30:00

If necessary, the day is adjusted to be valid for the resulting month.

HDIFF: Finding the Number of Units Between Two Date-Time Values

Use the following syntax in an expression to find the number of boundaries of a given type crossed in going from date 2 to date 1.

```
HDIFF (dtfield1, dtfield2, 'component', 'Dformat')
```

where:

dtfield1

Is the ending date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

dtfield2

Is the starting date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be used in the calculation, enclosed in single quotation marks. See [Component Names and Values for Use With Date-Time Functions](#) for a list of supported components. If the unit is weeks, the WEEKFIRST setting is used in the calculation.

Dformat

Is the USAGE format of the resulting number of units, enclosed in single quotation marks. The format type must be D.

Example Finding the Number of Days Between Two Date-Time Fields

The following request finds the number of days between the ADD_MONTH and TRANSDATE fields:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
DIFF_DAYS/D12.2 = HDIFF(ADD_MONTH, TRANSDATE, 'DAY', 'D12.2');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ADD_MONTH	DIFF_DAYS
-----	-----	-----	-----
1118	2000/06/26 05:45	2000/08/26 05:45:00	61.00
1237	2000/02/05 03:30	2000/04/05 03:30:00	60.00

HNAME: Extracting a Date-Time Component in Alphanumeric Format

Use the following syntax in an expression to extract a specified component from a date-time field and return it in alphanumeric format.

```
HNAME (dtfield, 'component', Aformat)
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be extracted, enclosed in single quotation marks. See [Component Names and Values for Use With Date-Time Functions](#) for a list of supported components.

Aformat

Is the alphanumeric USAGE format of the returned component, enclosed in single quotation marks. All other components are converted to strings of digits only. The year is always four digits, and the hour assumes the 24-hour system.

Example Extracting the Day Component in Alphanumeric Format From a Date-Time Field

The following request extracts the day in alphanumeric format from the TRANSDATE field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/A2 = HNAME(TRANSDATE, 'DAY', 'A2');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	DAY_COMPONENT
-----	-----	-----
1118	2000/06/26 05:45	26
1237	2000/02/05 03:30	05

Example Extracting the Week Component With Different WEEKFIRST Settings

The following request extracts the week in alphanumeric format from the TRANSDATE field. Changing the WEEKFIRST setting changes the value of the extracted component:

```
SET WEEKFIRST = 7
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
WEEK_COMPONENT/A10 = HNAME(TRANSDATE, 'WEEK', 'A10');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	WEEK_COMPONENT
-----	-----	-----
1118	2000/06/26 05:45	26
1237	2000/02/05 03:30	06

Running the same request setting WEEKFIRST to 3 produces the following output (see [How to Specify the First Day of the Week](#)):

CUSTID	DATE-TIME	WEEK_COMPONENT
-----	-----	-----
1118	2000/06/26 05:45	25
1237	2000/02/05 03:30	05

HPART: Extracting a Date-Time Component in Numeric Format

Use the following syntax in an expression to extract a specified component from a date-time field and return it in numeric format.

```
HPART (dtfield, 'component', 'Iformat')
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be extracted, enclosed in single quotation marks. See [Component Names and Values for Use With Date-Time Functions](#) for a list of supported components.

Iformat

Is the integer USAGE format of the returned component, enclosed in single quotation marks. The year is always four digits, and the hour assumes the 24-hour system.

Example Extracting the Day Component in Numeric Format From a Date-Time Field

The following request extracts the day in integer format from the TRANSDATE field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/I2 = HPART(TRANSDATE, 'DAY', 'I2');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	DAY_COMPONENT
-----	-----	-----
1118	2000/06/26 05:45	26
1237	2000/02/05 03:30	5

HSETPT: Inserting a Component Into a Date-Time Field

Use the following syntax in an expression to insert the numeric value of a specified component into a date-time field.

```
HSETPT (dtfield, 'component', value, {8|10}, 'Hformat')
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be inserted, enclosed in single quotation marks. See [Component Names and Values for Use With Date-Time Functions](#) for a list of supported components.

value

Is the numeric value to use for the requested component. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

Hformat

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Inserting the Day Component Into a Date-Time Field

The following request inserts the day into the ADD_MONTH field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
INSERT_DAY/HYYMDS = HSETPT(ADD_MONTH, 'DAY', 28, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ADD_MONTH	INSERT_DAY
-----	-----	-----	-----
1118	2000/06/26 05:45	2000/08/26 05:45:00	2000/08/28 05:45:00
1237	2000/02/05 03:30	2000/04/05 03:30:00	2000/04/28 03:30:00

HMIDNT: Setting the Time Portion of a Date-Time Field to Midnight

Use the following syntax in an expression to change the time portion of a date-time field to midnight (all zeroes). This function can be used for testing date-time fields for a given date.

```
HMIDNT (dtfield, {8|10}, 'Hformat')
```

where:

dtfield

Is date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

Hformat

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Setting the Time to Midnight

The following request sets the time portion of the TRANSDATE field to midnight in both the 24- and 12-hour systems:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_MID_24/HYYMDS = HMIDNT(TRANSDATE, 8, 'HYYMDS');
TRANSDATE_MID_12/HYYMDSA = HMIDNT(TRANSDATE, 8, 'HYYMDSA');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	TRANSDATE_MID_24	TRANSDATE_MID_12
1118	2000/06/26 05:45	2000/06/26 00:00:00	2000/06/26 12:00:00AM
1237	2000/02/05 03:30	2000/02/05 00:00:00	2000/02/05 12:00:00AM

HDATE: Converting the Date Portion of a Date-Time Field to a Date Format

Use the following syntax in an expression to extract the date portion of a date-time field and convert it to a date format (number of days since the base date 1900/12/31).

```
HDATE (dtfield, 'dateformat')
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

dateformat

Is the USAGE format of the returned date field (for example, YMD), enclosed in single quotation marks.

Example Converting the Date Portion of the TRANSDATE Field to a Date Format

The following request converts the date portion of the TRANSDATE field to date format YYMD:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	TRANSDATE_DATE
-----	-----	-----
1118	2000/06/26 05:45	2000/06/26
1237	2000/02/05 03:30	2000/02/05

HDTTM: Converting a Date field to a Date-Time Field

Use the following syntax in an expression to convert a date field to a date-time field. The time portion is set to midnight.

```
HDTTM (datefield, {8|10}, Hformat)
```

where:

datefield

Is the date value to be converted. You can supply the name of a date field, a date constant, or an expression that returns a date value.

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

Hformat

Is the USAGE format of the returned date-time value.

Example Converting a Date Field to a Date-Time Field

The following request converts the date field `TRANSDATE_DATE` to a date-time field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
DT2/HYYMDIA = HDTTM(TRANSDATE_DATE, 8, 'HYYMDIA');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	TRANSDATE_DATE	DT2
-----	-----	-----	---
1118	2000/06/26 05:45	2000/06/26	2000/06/26 12:00AM
1237	2000/02/05 03:30	2000/02/05	2000/02/05 12:00AM

HTIME: Converting the Time Portion of a Date-Time Field to a Number

Use the following syntax in an expression to convert the time portion of a date-time field to a numeric number of milliseconds (if the first argument is 8) or microseconds (if the first argument is 10). For microseconds, the input date-time field must be a 10-byte field.

```
HTIME ({8|10}, dtfield, 'Dformat')
```

where:

8 | 10

Is the length of the input date-time value. Use 8 for time values down to milliseconds, 10 for input time values down to microseconds.

dtfield

Is the date-time value to use for extracting the time. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

Dformat

Is the USAGE format of the returned number of milliseconds or microseconds, enclosed in single quotation marks.

Example Converting the Time Portion of a Date-Time Field to a Number

The following request converts time portion of the TRANSDATE field to a number of milliseconds:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
MILLISEC/D12.2 = HTIME(8, TRANSDATE, 'D12.2');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	MILLISEC
-----	-----	-----
1118	2000/06/26 05:45	20,700,000.00
1237	2000/02/05 03:30	12,600,000.00

HGETC: Storing the Current Date and Time in a Date-Time Field

Use the following syntax in an expression store the current date and time in a date-time field. If millisecond or microsecond values are not available in your operating environment, the value returned for these components is zero.

```
HGETC ({8|10}, 'Hformat')
```

where:

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for input time values down to microseconds.

Hformat

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

Example Storing the Current Date and Time in a Date-Time Field

The following request stores the current date and time in field DT2:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DT2/HYYMDm = HGETC(10, 'HYYMDm');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	DT2
-----	-----	---
1118	2000/06/26 05:45	2000/10/03 15:34:24.000000
1237	2000/02/05 03:30	2000/10/03 15:34:24.000000

NF755: Using FILEDEF for Creating Extract Files

By default in VM/CMS, extract files are written to the VM minidisk specified by the SET TEMP command. If you do not issue the SET TEMP command, extract files are written to the minidisk with the largest amount of unused space to which you have write access. The name of an extract file is the AS name specified in the command that creates it or, if no AS name is specified, a default name (HOLD, SAVE, or SAVB). The file type is assigned based on the extract file format.

This feature enables you to use a FILEDEF command to assign a file name, file type, and file mode for an extract file. In prior releases, FILEDEF was supported for creating SAVE and SAVB files, but not for creating HOLD files.

In the case of a HOLD file, the Master File is not affected by the FILEDEF command. The Master File is written to the minidisk specified by the SET TEMP command, and its name is taken from the AS name in the HOLD command. If the HOLD command does not contain an AS phrase, the Master File name is HOLD.

The HOLD, SAVE, or SAVB command can be issued within a request or after a request executes.

Note: The FILEDEF command is not supported on OS/390 or MVS. As in prior releases, you can use the DYNAM ALLOCATE or TSO ALLOCATE command to dynamically allocate extract files.

Syntax How to Use a FILEDEF Command for Creating an Extract File

Issue the following command before creating the extract file:

```
CMS FILEDEF ddname DISK filename filetype filemode
```

where:

ddname

Is the AS name from the HOLD, SAVE, or SAVB command. If the command did not specify an AS name, the ddname is HOLD, SAVE, or SAVB.

filename

Is the file name for the extract file.

filetype

Is the file type for the extract file.

filemode

Is the file mode for the extract file. You must have write access to this minidisk. If you do not have write access, the following error message is returned:

```
(FOC350) ERROR WRITING OUTPUT FILE: filename
```

Note:

- If a FOCUSORT file is created, it is written to the minidisk specified by the SET TEMP command.
- The FILEDEF command must be in effect any time you use FOCUS to access the file.
- Do not specify DCB parameters for a HOLD file; if you do, they will be ignored.
- The FILEDEF command is not supported for creating extract files in FOCUS format or other DBMS formats.
- The FILEDEF command is supported for MATCH FILE requests.

Example Using FILEDEF to Create a HOLD File

In the following examples, SET TEMP = T. The request is:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY LAST_NAME BY FIRST_NAME
ON TABLE HOLD AS CURRSAL
END
```

Running this request with no FILEDEF command creates the following files:

```
CURRSAL MASTER T1
CURRSAL FOCTEMP T1
```

Issue the following FILEDEF command:

```
CMS FILEDEF CURRSAL DISK SALLIST DATA A
```

Now, running the same request creates the following files:

```
CURRSAL MASTER T1
SALLIST DATA A1
```

Note that the file name, file type, and destination minidisk for the extract file are taken from the FILEDEF command. The file name and destination minidisk for the Master File are not.

NF759: Increased Number of Display Fields

The maximum number of display fields allowed in a request has been increased from 256 to approximately 495. This number includes all named fields, whether printed or not, such as data source fields, temporary fields (virtual fields and calculated values), certain internal fields (for example, TABPAGENO) and fields used in headings and footings. The increased field limit is the maximum number of fields allowed when each field has the smallest length possible (A4 ACTUAL). Longer field lengths reduce the total number of printable fields..

Reference Error Messages for Number of Display Fields

(FOC005) THE NUMBER OF VERB OBJECTS EXCEEDS THE MAXIMUM

Up to 495 fields may be used in a single report request. This total does not include sort fields.

(FOC098) TOTAL LENGTH OF VERB OBJECTS EXCEED 3956 CHARACTERS

The length of all verb objects was greater than 3956. Adjust the lengths or quantity of fields and reissue the TABLE request.

Reference Usage Notes for Number of Display Fields

- Out of 4K there exists a 3956 byte data area. (The data area is where fields are stored for report creation.)
- Every field in this data area is rounded up to a full word boundary.
- Every field gets a four-byte counter field associated with it and the four-byte counter field affects the total number of bytes in this data area.
- Field Prefixes, and formatting options may also affect the available data area.

NF761: Comma Suppress Edit Format Option

The comma suppress edit format option allows you to suppress the display of commas. This gives you the ability to display numeric and monetary data without commas and gives you an additional display option when using COMPUTE/DEFINE.

Syntax How to Suppress Commas

The comma suppress edit option is invoked by including a lower case 'c' after numeric format options M and N (floating and non-floating dollar sign) and data format D (floating-point double-precision).

Format	Data	Display
D6c	41376	41376
D7Mc	6148	\$6148
D7Nc	6148	\$ 6148

Reference Error Messages for Comma Suppress Option

Using the comma suppress edit option with alphanumeric data types causes the following error message to be displayed:

`(FOC207) ERROR IN THE FORMAT DEFINITION OF FIELD`

Reference Considerations for Comma Suppress Option

This feature may only be used with numeric format options that automatically invoke comma inclusion option C.

Using this option with other numeric data types has no effect.

NF762: Percent Edit Format Option

FOCUS provides a format option that enables you to display a percent sign along with numeric data. This allows you to display data as percentages after calculations using COMPUTE/DEFINE.

Syntax How to Display Percent Signs

The percent edit option is invoked by including a '%' after any numeric format option.

Format	Data	Display
I2%	21	21%
D7%	6148	6,148%
F3.2%	48	48.00%

Reference Error Messages for Percent Display Option

Using the percent edit format option with either alphanumeric data types, format options M and N, or date formats results in the following error message being displayed:

(FOC207) ERROR IN THE FORMAT DEFINITION OF FIELD

Reference Special Considerations for Percent Display Option

This feature may only be used with numeric data.

This feature does not perform calculations intended to derive a percentage of data. It is only a format option that allows numeric data to be displayed as a percentage.

The percent sign may only be displayed on the right hand side of the data.

NF766: DEFINE Functions

A DEFINE function is a named group of calculations that use any number of input values and produce a return value. These functions are not tied to a specific Master File or request and therefore can be reused in different contexts throughout your FOCUS session.

Using DEFINE Functions

A DEFINE function can be called in the same situations that are valid for user-written subroutines. Data types (numeric or alphanumeric) must match between the arguments defined and the arguments used. Shorter alphanumeric arguments are padded with blanks while longer alphanumeric arguments are truncated.

All calculations within the function are done in double precision. As with other FOCUS calculations, format conversions occur only across equal signs in the assignments that define temporary fields.

Before calling a DEFINE function, you must issue the commands that define the function at the FOCUS prompt or in a stored procedure.

Syntax How To Define a Function

```
DEFINE FUNCTION name (parameter1/format1,..., parametern/formatn)
[tempvariablea/formata = expressiona;]
.
.
.
[tempvariablex/formatx = expression,x]
name/format = [result_expression];
END
```

where:

name

Is the name of the function. This must be the last field calculated in the function and is used to return the value of the function to the calling procedure.

format

Is the format of the value the function returns.

parameter1/format1...parametern/formatn

Are the parameter names and their formats.

If a parameter is alphanumeric, the calling argument must be alphanumeric. Shorter calling arguments are padded on the right with blanks, and longer arguments are truncated.

If a parameter is numeric, the calling argument must also be numeric. To prevent unexpected results, you must be consistent in your use of data types.

tempvariablea/formata...tempvariablex/formatx

Are temporary fields and their formats. Temporary fields hold intermediate values used in the function. You can define as many temporary fields as you need.

expressiona...expressionx

Are the expressions that calculate the temporary field values. The expressions can use parameters, constants, and other temporary fields defined in the same function.

result_expression

Is the expression that calculates the value returned by the function. The expression can use parameters, constants, and temporary fields defined in the same function.

All names defined in the body of the function are local to the function. The last field defined before the END command in the function definition must have the same name as the function and represents the return value for the function.

Example Defining a Function

```
DEFINE FUNCTION SUBTRACT (VAL1/D8, VAL2/D8)
SUBTRACT/D8.2 = VAL1 - VAL2;
END
```

```
TABLE FILE MOVIES
PRINT TITLE LISTPR IN 35 WHOLESALPR AND COMPUTE
PROFIT/D8.2 = SUBTRACT(LISTPR,WHOLESALPR);
BY CATEGORY
  WHERE CATEGORY EQ 'MYSTERY' OR 'ACTION'
END
```

SUBTRACT is the name of the function. It uses local parameters VAL1 and VAL2.

The output is:

CATEGORY	TITLE	LISTPR	WHOLESALEPR	PROFIT
-----	-----	-----	-----	-----
ACTION	JAWS	19.95	10.99	8.96
	ROBOCOP	19.98	11.50	8.48
	TOTAL RECALL	19.99	11.99	8.00
	TOP GUN	14.95	9.99	4.96
	RAMBO III	19.95	10.99	8.96
MYSTERY	REAR WINDOW	19.98	9.00	10.98
	VERTIGO	19.98	9.00	10.98
	FATAL ATTRACTION	29.98	15.99	13.99
	NORTH BY NORTHWEST	19.98	9.00	10.98
	DEAD RINGERS	25.99	15.99	10.00
	MORNING AFTER, THE	19.95	9.99	9.96
	PSYCHO	19.98	9.00	10.98
	BIRDS, THE	19.98	9.00	10.98
	SEA OF LOVE	59.99	30.00	29.99

Syntax How to Query DEFINE Functions

You can display a list of all defined functions and their parameters by issuing:

```
? FUNCTION
```

A screen similar to the following is displayed:

```
FUNCTIONS CURRENTLY ACTIVE
```

Name	Format	Parameter	Format
-----	-----	-----	-----
SUBTRACT	D8.2	VAL1	D8
		VAL2	D8

If you issue the ? FUNCTION command with no functions defined, the following is displayed:

```
NO FUNCTIONS CURRENTLY IN EFFECT
```


Syntax How to Clear DEFINE Functions

You can clear functions by issuing:

```
DEFINE FUNCTION {name | *} CLEAR
```

where:

name

Is the function name to clear.

*

Clears all active functions.

Reference DEFINE Function Limits and Restrictions

- The number of functions you can define and use in a session is virtually unlimited.
- DEFINE functions are not cleared by issuing a JOIN, or any other FOCUS command, with the exception of DEFINE FUNCTION CLEAR.
- Function names are limited to eight characters. There is no limit on the number of parameters.
- Parameter names are limited to twelve characters.
- DEFINE functions can use other DEFINE functions but cannot be used recursively.
- If you overwrite or clear a DEFINE function, a subsequent attempt to use a temporary field that refers to that function will generate the following message:

```
(FOC1956) DEFINE FUNCTION %1 used after CLEAR or reDEFINE
```

```
DEFINE FUNCTION dfname was invoked, but it no longer exists.  
A "DEFINE FUNCTION dfname" command was processed  
subsequently (with or without CLEAR), thereby invalidating any  
references to the the original version of dfname.
```

Reference DEFINE FUNCTION Error Messages

- (FOC1940) **DEFINE FUNCTION name not 1-8 characters: %1**
The name given for a DEFINE FUNCTION (the character string following "DEFINE FUNCTION") contains more than eight characters or is null.
- (FOC1941) **DEFINE FUNCTION %1 CLEAR--illegal delimiter**
In a "DEFINE FUNCTION dfname CLEAR" command, the character string dfname contains a forward slash, an open or close parenthesis, or a comma; this is not supported.
- (FOC1942) **DEFINE FUNCTION not found: %1**
In a "DEFINE FUNCTION dfname CLEAR" command, the syntax is proper but dfname does not exist, so it cannot be CLEARed.
- (FOC1943) **DEFINE FUNCTION needs close paren to delimit params: %1**
A DEFINE FUNCTION definition must contain a list of parameters delimited by a close parenthesis. The DEFINE FUNCTION parser encountered END or a similar termination without finding a close parenthesis.
- (FOC1944) **No open paren after DEFINE FUNCTION format: %1**
A format may be specified for a DEFINE FUNCTION immediately after the DEFINE FUNCTION name (and a forward slash) and before the open parenthesis enclosing the parameter list. The forward slash is coded but the next delimiter is not an open parenthesis.

- (FOC1945) `DEFINE FUNCTION format not 1-8 characters: %1`
A format may be specified for a DEFINE FUNCTION immediately after the DEFINE FUNCTION name (and a forward slash) and before the open parenthesis enclosing the parameter list. The length of the format specification must be from one to eight characters.
- (FOC1946) `Name of DEFINE FUNCTION not delimited by "/" or "(": %1`
The delimiter for the DEFINE FUNCTION name must be a forward slash (if a format specification follows) or an open parenthesis (if there is no format specification). The actual delimiter is neither of these.
- (FOC1947) `DEFINE FUNCTION parameter name not delimited by "/": %1`
A format specification must follow each parameter for a DEFINE FUNCTION, and the delimiter that separates the name from the format specification must be a forward slash. The actual delimiter is not a forward slash.
- (FOC1948) `DEFINE FUNCTION parameter name not 1-12 characters: %1`
The name of each parameter for a DEFINE FUNCTION must contain from one to twelve characters. Either the parameter name shown is null or it contains more than twelve characters.
- (FOC1949) `DEFINE FUNCTION parameter format not delimited by ", " or ")": %1`
A format specification must follow each parameter for a DEFINE FUNCTION, and the delimiter that marks the end of the specification must be a comma (if there are additional parameters) or a close parenthesis (if this is the last parameter).

- (FOC1950) `DEFINE FUNCTION parameter format not 1-8 characters: %1`
A format specification must follow each parameter for a DEFINE FUNCTION, and it must contain from one to eight characters. Either the actual specification contains more than eight characters or it is null.
- (FOC1951) `Invalid DEFINE FUNCTION parameter format: %1`
The format specification given for the parameter shown for a DEFINE FUNCTION is not recognized.
- (FOC1952) `DEFINE FUNCTION %1 is void`
At least one computational statement must appear between the parameter list for DEFINE FUNCTION and "END", but the actual DEFINE FUNCTION has no computational statements.
- (FOC1953) `DEFINE FUNCTION name %1 not equal to final assignment name %2`
The name immediately following "DEFINE FUNCTION" must match the name on the left-hand side of the final computational statement given for the DEFINE FUNCTION, but it does not match.
- (FOC1954) `DEFINE FUNCTION format %1 not equal to final assignment format %2`
The (optional) format specified before the parameters for a DEFINE FUNCTION must match the format on the left-hand side of the final computational statement given for the DEFINE FUNCTION, but it does not match.
- (FOC1955) `DEFINE FUNCTION %1 entered recursively`
A DEFINE FUNCTION was invoked, but before returning it was invoked again; this is not supported. Note that the invalid recursive usage may have been indirect: DF1 invokes DF2, which invokes DF3, ... , which invokes DF1.

- (FOC1956) **DEFINE FUNCTION %1 used after CLEAR or reDEFINE**
DEFINE FUNCTION dfname was invoked, but it no longer exists. A "DEFINE FUNCTION dfname" command was processed subsequently (with or without CLEAR), thereby invalidating any references to the original version of dfname.
- (FOC1957) **Warning--DEFINE FUNCTION %1 superseded; could result in FOC1956**
A successful DEFINE FUNCTION dfname has replaced the prior dfname. Trying to use a DEFINE FILE (or another DEFINE FUNCTION) that refers to the prior version of dfname will result in FOC1956. Such a DEFINE FILE/FUNCTION could be re-issued, which would direct it to use the new version of dfname.
- (FOC1958) **DEFINE FUNCTION %1 takes %2 parameter(s), but # supplied**
DEFINE FUNCTION dfn requires Nexp input parameters, but Nsupp were supplied. dfn, Nexp, and Nsupp are displayed within the message text.

NF773: Token Delimited Files

FOCUS can now read single-segment sequential data sources in which fields are separated by any delimiter.

Syntax How to Define Files With Delimiters

Delimiters must be defined in the Master File. The FILE declaration must include the following attribute:

```
SUFFIX=DFIX
```

To use a delimiter that consists of a single non-printable character or of one or more printable characters, the delimiter is defined as a field with the following attributes:

```
FIELDNAME=DELIMITER, ALIAS=delimiter, USAGE=ufmt, ACTUAL=afmt , $
```

To use a delimiter that consists of multiple non-printable characters or a combination of printable and non-printable characters, the delimiter is defined as a group:

```
GROUP=DELIMITER, ALIAS= , USAGE=ufmtg, ACTUAL=afmtg , $  
FIELDNAME=DELIMITER, ALIAS=delimiter1, USAGE=ufmt1, ACTUAL=afmt1 , $  
.  
.  
.  
FIELDNAME=DELIMITER, ALIAS=delimitern, USAGE=ufmtn, ACTUAL=afmtn , $
```

where:

```
DELIMITER
```

Indicates that the field or group is used as the delimiter in the data source.

delimiter

Identifies a delimiter.

For one or more printable characters, the value consists of the actual characters. The delimiter must be enclosed in single quotation marks if it includes characters used as delimiters in Master File syntax.

For a non-printable character, the value is the decimal equivalent of the EBCDIC or ASCII representation of the character, depending on your operating environment.

ufmt, afmt

Are the USAGE and ACTUAL formats for the delimiter. Possible values are:

Type of delimiter	USAGE	ACTUAL
Printable characters	<i>An</i> where <i>n</i> is the number of characters	<i>An</i> where <i>n</i> is the number of characters
Non-printable character such as Tab	<i>I4</i>	<i>I1</i>
Group (combination of printable and non-printable characters, or multiple non-printable characters)	Sum of the individual USAGE lengths	Sum of the individual ACTUAL lengths

Reference Usage Notes for Token Delimited Files

- If the delimiter is alphanumeric and the delimiter value contains special characters (those used as delimiters in Master File syntax), it must be enclosed in single quotation marks.

- Numeric (decimal) values may be used to represent any character, but are predominantly used for non-printable characters such as Tab. The numeric values may differ between EBCDIC and ASCII platforms.
- A delimiter is needed to separate field values. A pair of delimiters denotes a missing or default field value.
- Trailing delimiters are not necessary except that all fields must be terminated with the delimiter if the file resides in CMS or has fixed length records in MVS.
- Only one delimiter field/group is permitted per Master File.
- Token delimited files cannot be used in joins.

Example Defining Delimiters

The following example shows a one-character alphanumeric delimiter:

```
FIELDNAME=DELIMITER, ALIAS=' ' ,USAGE=A1, ACTUAL=A1 , $
```

The following example shows a two-character alphanumeric delimiter:

```
FIELDNAME=DELIMITER, ALIAS>// ,USAGE=A2, ACTUAL=A2 , $
```

The following example shows how to use the Tab character as a delimiter:

```
FIELDNAME=DELIMITER, ALIAS=05 ,USAGE=I4, ACTUAL=I1 , $
```

The following example shows how to use a blank character described as a numeric delimiter:

```
FIELDNAME=DELIMITER, ALIAS=64 ,USAGE=I4, ACTUAL=I1 , $
```

The following example shows a group delimiter (Tab-slash-Tab combination):

```
GROUP=DELIMITER, ALIAS= ,USAGE=A9, ACTUAL=A3 , $  
FIELDNAME=DEL1, ALIAS=05 ,USAGE=I4, ACTUAL=I1 , $  
FIELDNAME=DEL2, ALIAS=/ ,USAGE=A1, ACTUAL=A1 , $  
FIELDNAME=DEL3, ALIAS=05 ,USAGE=I4, ACTUAL=I1 , $
```


Example Separating Field Values for Missing Data

The following Master File shows the MISSING attribute specified for the CAR field:

```
FILE=DFIXF01 ,SUFFIX=DFIX
SEGNAME=SEG1 ,SEGTYPE=S0
FIELDNAME=COUNTRY ,ALIAS=F1 ,USAGE=A10 ,ACTUAL=A10 ,,$
FIELDNAME=CAR ,ALIAS=F2 ,USAGE=A16 ,ACTUAL=A16 ,MISSING=ON, $
FIELDNAME=NUMBER ,ALIAS=F3 ,USAGE=P10 ,ACTUAL=Z10 ,,$
FIELDNAME=DELIMITER ,ALIAS=' , ' ,USAGE=A1 ,ACTUAL=A1 ,,$
```

In the source file, two consecutive comma delimiters indicate missing values for CAR:

```
GERMANY ,VOLKSWAGEN ,1111
GERMANY ,BMW ,
USA ,CADILLAC ,22222
USA ,FORD
USA , ,44444
JAPAN
ENGLAND ,
FRANCE
```

The output is:

<u>COUNTRY</u>	<u>CAR</u>	<u>NUMBER</u>
GERMANY	VOLKSWAGEN	1111
GERMANY	BMW	0
USA	CADILLAC	22222
USA	FORD	0
USA	.	44444
JAPAN	.	0
ENGLAND	.	0
FRANCE	.	0

NF777: Two-Gigabyte FOCUS Database Support

The FOCUS database size has been increased to a maximum of two gigabytes per physical data file, overcoming previous FOCUS database size limitations. Through partitioning, one logical FOCUS database can now span up to 500 gigabytes. FOCUS database size was previously limited to 64 gigabytes through the use of LOCATION files.

Note that the discussion in this document applies to any FOCUS file that has extended beyond one-gigabyte but has not reached the two-gigabyte limit.

For information about partitioning FOCUS data sources, see [NF777: Partitioned FOCUS Data Sources](#).

Enabling Two-Gigabyte Support

In order to enable support for two-gigabyte databases, you need to set the value of the FOC2GIGDB parameter to ON in the FOCPARM profile.

Syntax How to Enable Two-Gigabyte Support

Issue the following command in the FOCPARM profile:

```
SET FOC2GIGDB = {ON|OFF}
```

where:

ON

Enables support for FOCUS data sources larger than one-gigabyte. Note that an attempt to use FOCUS data sources larger than one-gigabyte in a release prior to FOCUS Version 7.1 can cause database corruption.

OFF

Disables support for FOCUS data sources larger than one-gigabyte. OFF is the default value.

Reference Usage Notes for Two-Gigabyte FOCUS Data Sources

- To sort a FOCUS data source that is larger than one-gigabyte, on MVS you must explicitly allocate ddname FOCSORT to a temporary file with enough space to contain the data; on VM, you must have enough TEMP space available.
- To REBUILD a FOCUS data source that is larger than one-gigabyte, on MVS you must explicitly allocate ddname REBUILD to a temporary file with enough space to contain the data; on VM you must have enough TEMP space available. It is strongly recommended that you REBUILD/REORG to a new file, in sections, to avoid the need to allocate large amounts of space to REBUILD. In the DUMP phase, use selection criteria to dump a section of the database. In the LOAD phase, make sure to *add* each new section after the first. To add to a database in MVS you must issue the LOAD command with the following syntax:

LOAD NOCREATE

- If you create a FOCUS data source that is larger than one gigabyte using HOLD FORMAT FOCUS, on MVS you must explicitly allocate ddnames FOC\$HOLD and FOCSORT to temporary files large enough to hold the data; on VM you must have enough TEMP space available.

NF777: Partitioned FOCUS Data Sources

Through partitioning, one logical FOCUS database can now span up to 500 gigabytes. FOCUS database size was previously limited to 64 gigabytes through the use of LOCATION files.

For information about two-gigabyte support, see [NF777: Two-Gigabyte FOCUS Database Support](#)

Partitioning

FOCUS data sources can now consist of up to 250 physical files of up to two gigabytes each, for a maximum of 500 gigabytes of real storage per logical database. The new horizontal partition is a slice of the entire file structure, meaning that new FOCUS database partitions are not subject to the multiplicative effect of LOCATION-style vertical partitioning. Note, however, that the number of physical files associated with one FOCUS data source is the sum of all of its partitions and LOCATION files. This sum must be less than or equal to 250. New FOCUS data sources can grow in size over time, and can be repartitioned based on the requirements of the application.

Note: You do not have to partition your data source. If you choose not to, your application will automatically support the increased FOCUS database size when you set the FOC2GIGDB parameter to ON.

Intelligent Partitioning

The FOCUS database now supports intelligent partitioning, which means that each horizontal partition contains the complete database structure for specific data values or ranges of values. Intelligent partitioning lets you not only separate the data into up to 250 physical two-gigabyte files, it allows you to create an Access File in which you describe, using WHERE criteria, the actual data values in each partition. When processing a report request, the selection criteria in the request are compared to the WHERE criteria in the Access File to determine which partitions are required for retrieval.

To select applications that can benefit most from partitioning, look for applications that employ USE commands to concatenate data sources or for data that lends itself to separation based on data values or ranges of values, such as data stored by month or by department. Intelligent partitioning functions like an intelligent USE. It looks at the Access File when processing a report request to determine which partitions to read, whereas the USE command reads all of the files on the list. This intelligence decreases I/O and delivers significant performance benefits.

To take advantage of the partitioning feature, you must:

- Edit the Master File and add the ACCESSFILE attribute.
- Create the Access File using a text editor.

Concatenation of multiple partitions is supported for reporting only. You must load or rebuild each physical partition separately. You can either create a separate Master File for each partition to reference in the load procedure, or you can use the single Master File created for reporting against the partitioned data source, if you:

- Issue an explicit allocation command to link the Master File to each partition in turn.
- Run the load procedure for each partition in turn.

Note: Report requests will automatically read all required partitions without user intervention.

Specifying an Access File in a FOCUS Master File

To take advantage of the partitioning feature, you must edit the Master File and add the ACCESSFILE attribute to identify the name of the Access File.

Syntax How to Specify an Access File for a Partitioned FOCUS Database

```
FILENAME=fname, SUFFIX=FOC, ACCESS[FILE]=accessfile,  
.  
.  
.
```

where:

fname

Is the file name of the partitioned data source.

accessfile

Is the name of the Access File. Note that this can be any valid name.

Example Master File for the VIDEOTR2 Partitioned Database

```
FILENAME=VIDEOTR2,  SUFFIX=FOC,
ACCESS=VIDEOACX,  $
SEGNAME=CUST,      SEGTYPE=S1
  FIELDNAME=CUSTID,    ALIAS=CIN,          FORMAT=A4,      $
  FIELDNAME=LASTNAME,  ALIAS=LN,          FORMAT=A15,     $
  FIELDNAME=FIRSTNAME, ALIAS=FN,          FORMAT=A10,     $
  FIELDNAME=EXPDATE,   ALIAS=EXDAT,        FORMAT=YMD,     $
  FIELDNAME=PHONE,     ALIAS=TEL,          FORMAT=A10,     $
  FIELDNAME=STREET,    ALIAS=STR,          FORMAT=A20,     $
  FIELDNAME=CITY,      ALIAS=CITY,         FORMAT=A20,     $
  FIELDNAME=STATE,     ALIAS=PROV,         FORMAT=A4,      $
  FIELDNAME=ZIP,       ALIAS=POSTAL_CODE,  FORMAT=A9,      $
SEGNAME=TRANSDAT,  SEGTYPE=SH1,  PARENT=CUST
  FIELDNAME=TRANSDATE, ALIAS=OUTDATE,   FORMAT=HYMDI,   $
SEGNAME=SALES,     SEGTYPE=S2,  PARENT=TRANSDAT
  FIELDNAME=TRANSCODE, ALIAS=TCOD,       FORMAT=I3,      $
  FIELDNAME=QUANTITY,  ALIAS=NO,         FORMAT=I3S,     $
  FIELDNAME=TRANSTOT,  ALIAS=TTOT,       FORMAT=F7.2S,   $
SEGNAME=RENTALS,  SEGTYPE=S2,  PARENT=TRANSDAT
  FIELDNAME=MOVIECODE, ALIAS=MCOD,       FORMAT=A6,  INDEX=I, $
  FIELDNAME=COPY,      ALIAS=COPY,       FORMAT=I2,      $
  FIELDNAME=RETURNDATE, ALIAS=INDATE,     FORMAT=YMD,     $
  FIELDNAME=FEE,       ALIAS=FEE,         FORMAT=F5.2S,   $
DEFINE DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
```

The FOCUS Access File

The Access File provides comprehensive metadata management for all FOCUS data sources. It shields end users from the complex file storage and configuration details used for efficient and transparent access to partitioned and distributed databases.

The Access File describes how to locate, concatenate, join, and select the appropriate physical data files for retrieval requests against one or more FOCUS databases. Access Files are optional in retrieval requests against non-partitioned databases with no location files and play no part in data maintenance requests.

Every request supplies the name of a Master File. The Master File is read and the declarations in it are used to access the data source. If the Master File includes an ACCESSFILE attribute, FOCUS reads the named Access File and uses it to locate the correct data files. Each Master File can point to its own separate Access File, or several Master Files can point to the same Access File. This flexibility makes it possible to create one Access File that manages database access for an entire application. If the Master File does not contain an ACCESSFILE attribute, FOCUS attempts to satisfy the request with the Master File alone.

You can use an Access File to take advantage of the following database features:

- Horizontal and vertical partitioning. A database can consist of several separate files, or horizontal partitions, each of which contains the database records for a specific time period, region, or other element. Segments can also be stored separately from the rest of the data source (LOCATION files or vertical partitions). The Access File describes how to concatenate the separate data files.
- Joins. If joined files are partitioned, the Access File describes how to concatenate the separate data files in the join.

An Access File is *required* to take advantage of intelligent partitioning. Intelligent partitioning places specific data values in each physical partition and uses the Access File to describe the values in each partition. With this information, FOCUS optimizes database access by retrieving only those partitions whose values are consistent with the selection criteria in the request.

Note: On OS/390 the Access File must be a member of a data set concatenated in the allocation for ddname ACCESS. On VM/ESA the Access File must have the file type ACCESS. FOCUSQL cannot be used as the file type. The Access File has the same DCB attributes as the Master File (LRECL=80, RECFM=FB, BLKSIZE= multiple of LRECL).

FOCUS Access File Attributes

The Access File can include the following attributes:

Attribute	Synonyms	Description
MASTERNAME	MASTER	A Master File entry.
DATANAME	DATA	The name of the physical file.
WHERE		The WHERE criteria.
LOCATION		A segment location.

Each Access File declaration begins with a MASTERNAME attribute that identifies the Master File to which it applies. By including multiple MASTERNAME declarations, you can use one Access File for multiple Master Files, possibly for an entire application.

Syntax How to Create an Access File

```
MASTERNAME filename1
  DATANAME dataname1 [WHERE test1 ;]
    [LOCATION locationnamea DATANAME datanamea]
    .
    .
    .
  DATANAME dataname2 [WHERE test2 ;]
    [LOCATION locationnameb DATANAME datanameb]
    .
    .
    .
MASTERNAME filename2
  .
  .
  .
```

where:

MASTERNAME

Is the attribute that identifies the Master File name. MASTER is a synonym for MASTERNAME.

filename1, filename2

Are names of Master Files. You can describe unrelated Master Files in one Access File.

DATANAME

Is the attribute that identifies a physical file. DATA is a synonym for DATANAME.

dataname1, dataname2

Are the fully qualified physical file names of physical partition files, in the syntax native to your operating environment.

test

Is a valid WHERE test. The following types of expressions are supported. You can also combine any number of these expressions with the AND operator:

fieldname relational_operator value1 [OR value2 OR value3 ...]

fieldname FROM value1 TO value2 [OR value3 TO value4 ...]

fieldname1 FROM value1 TO value2 [OR fieldname2 FROM value3 TO value4 ...]

where:

fieldname, fieldname1, fieldname2

Are field names in the Master File.

relational_operator

Can be one of the following: EQ, NE, GT, GE, LT, LE.

value1, value2, value3, value4

Are valid values for their corresponding fields.

Note: If the test conditions do not accurately reflect the contents of the files, you may get incorrect results from requests.

LOCATION

Is the attribute that identifies a separately stored segment.

locationnamea, locationnameb

Are the values of the LOCATION attributes from the Master File.

Segment locations must map one-to-one to horizontal partitions.

datanamea, datanameb

Are the fully qualified physical file names of the LOCATION files, in the syntax native to your operating environment.

Example Describing Intelligent Partitions in a FOCUS Access File

The following Access File illustrates how to define intelligent partitions for the VIDEOTR2 database, in which data is grouped by date.

For MVS:

```
MASTERNAME VIDEOTR2
  DATANAME USER1.VIDPART1.FOCUS
    WHERE DATE EQ 1991;

  DATANAME USER1.VIDPART2.FOCUS
    WHERE DATE FROM 1996 TO 1998;

  DATANAME USER1.VIDPART3.FOCUS
    WHERE DATE FROM 1999 TO 2000;
```

For CMS:

```
MASTERNAME VIDEOTR2
  DATANAME 'VIDPART1 FOCUS A'
    WHERE DATE EQ 1991;

  DATANAME 'VIDPART2 FOCUS A'
    WHERE DATE FROM 1996 TO 1998;

  DATANAME 'VIDPART3 FOCUS A'
    WHERE DATE FROM 1999 TO 2000;
```

Example Describing Intelligent Partitions With LOCATION Files

Consider the following version of a SALES Master File. The CUSTDATA segment is stored in a separate LOCATION file named MORECUST:

```
FILENAME=SALES, ACCESSFILE=XYZ,$
  SEGNAME=SALEDATA
.
.
.
  SEGNAME=CUSTDATA, LOCATION=MORECUST,$
```

The corresponding Access File (XYZ) describes one partition for 1994 data, and another partition for the 1993 data. Each partition has its corresponding MORECUST LOCATION file:

For MVS:

```
MASTERNAME SALES
  DATANAME USER1.SALES94.FOCUS
    WHERE SDATE FROM '19940101' TO '19941231';
  LOCATION MORECUST
    DATANAME USER1.MORE1994.FOCUS

  DATANAME USER1.SALES93.FOCUS
    WHERE SDATE FROM '19930101' TO '19931231';
  LOCATION MORECUST
    DATANAME USER1.MORE1993.FOCUS
```

For CMS:

```
MASTERNAME SALES
  DATANAME 'SALES94 FOCUS A'
    WHERE SDATE FROM '19940101' TO '19941231';
  LOCATION MORECUST
    DATANAME 'MORE1994 FOCUS A'

  DATANAME 'SALES93 FOCUS A'
    WHERE SDATE FROM '19930101' TO '19931231';
  LOCATION MORECUST
    DATANAME 'MORE1993 FOCUS A'
```

Example Using a Partitioned Database

The following illustrates how to use a partitioned database.

```
TABLE FILE VIDEOTR2
PRINT LASTNAME FIRSTNAME DATE
WHERE DATE FROM 1996 TO 1997
END
```

The output is:

LASTNAME	FIRSTNAME	DATE
-----	-----	----
HANDLER	EVAN	1996
JOSEPH	JAMES	1997
HARRIS	JESSICA	1997
HARRIS	JESSICA	1996
MCMAHON	JOHN	1996
WU	MARTHA	1997
CHANG	ROBERT	1996

There is nothing in the request or output that signifies that a partitioned database was used. However, only the second partition is retrieved, reducing I/O and enhancing performance.

Describing Joined Files

The Master File can describe cross-references to other Master Files. In simple cases, the Master File alone may be sufficient for describing the cross-reference.

If one of the joined files is horizontally partitioned, only that file needs an Access File to implement the join.

However, when both of the joined files are horizontally partitioned, they can both be described in one Access File or they can each be described in a separate Access File in order to implement the join. Only the host file is allowed to have WHERE criteria in the Access File. If both the host and cross-referenced file have WHERE criteria, a join may produce unexpected results.

Example Joining Two Partitioned Data Sources

Recall that the cross-referenced field in a join must be indexed. If the host file is partitioned, the cross-referenced file must either contain the same number of partitions as the host file or only one partition.

For MVS:

```
MASTERNAME SALES
  DATANAME USER1.NESALES.FOCUS
  DATANAME USER1.MIDSALES.FOCUS
  DATANAME USER1.SOSALES.FOCUS
  DATANAME USER1.WESALES.FOCUS
```

```
MASTERNAME CUSTOMER
  DATANAME USER1.NECUST.FOCUS
  DATANAME USER1.MIDCUST.FOCUS
  DATANAME USER1.SOCUST.FOCUS
  DATANAME USER1.WECUST.FOCUS
```

For CMS:

```
MASTERNAME SALES
  DATANAME 'NESALES FOCUS A'
  DATANAME 'MIDSALES FOCUS A'
  DATANAME 'SOSALES FOCUS A'
  DATANAME 'WESALES FOCUS A'
```

```
MASTERNAME CUSTOMER
  DATANAME 'NECUST FOCUS A'
  DATANAME 'MIDCUST FOCUS A'
  DATANAME 'SOCUST FOCUS A'
  DATANAME 'WECUST FOCUS A'
```

Reference Usage Notes for Partitioned FOCUS Data Sources

- Concatenation of multiple partitions in one request is only valid for reporting. To MODIFY or REBUILD a partitioned database, you must explicitly allocate and MODIFY, Maintain, or REBUILD one partition at a time.
- The order of precedence for allocating data sources is as follows:
 - A USE command in effect has the highest precedence. It overrides an Access File or an explicit allocation for a data source.
 - An Access File overrides an explicit allocation for a data source.
- A DATASET attribute cannot be used in the same Master File as an ACCESSFILE attribute.

Reference Error Messages

(ACC20201) BOTH DATASET AND ACCESS FILE NOT PERMITTED

A Master file cannot specify a DATASET and an Access File at the same time.

- (ACC20202) MEMORY ALLOCATION ERROR
Access File Memory Failure.
- (ACC20203) ACCESS FILE %1 (MASTER %2) NOT FOUND
The Access File specified in the Master could not be found.
Check the spelling and/or the location of the Access File.
- (ACC20204) ACCESS FILE SYSTEM ERROR
The Access File system could not execute the Access File.
- (ACC20205) UNABLE TO OPEN ACCESS FILE %1
The Access File specified could not be opened. Check the permissions and/or the name of the Access File.
- (ACC20206) ACCESS FILE %1 HAS MULTIPLE ENTRIES FOR MASTER %2
No Access File may have two entries for a single master. Remove all duplicate entries from the Access File
- (ACC20207) NO ENTRY FOUND FOR MASTER %1 IN ACCESS FILE %2
An Access File specified in the Master must have a corresponding MASTER name. Add a MASTER entry to the Access File, or remove the Access File from the Master.
- (ACC20208) UNABLE TO PARSE ACCESS FILE
An error occurred while trying to parse the Access File.
- (ACC20209) PARSER ERROR: %1
There is a grammatical and/or syntactical error in the Access File. If nn/ii/ll is shown then look at line nn, column ii, for a length of ll for the problem token.
- (ACC20211) UNABLE TO PARSE WHERE CLAUSE
The WHERE clause in the Access File could not be parsed.
Check the clause and parse the Master again.

- (ACC20212) ACCESS FILE LOGICAL NAME LENGTH EXCEEDED : %1
Logical Names: Master File Name, External Location Names, and External Index Names Logical names may not exceed 8 characters.
- (ACC20215) ACCESS FILE SYSTEM ERROR FOR FILE : %1
Check the physical file name in the Access File.
- (ACC20216) TOO MANY PARTITIONS. PARTITION %1 FOR MASTER %2 EXCEEDS LIMIT.
There is a limit to the total number of physical partitions that may be used in a request. Delete partitions that are not needed from the Access File.
- (ACC20217) ACCESS FILE TOKEN %1 TOO LONG
The Token in the Access File is too long. No Token may be longer than 80 characters.
- (ACC20219) PARTITION %2 IS MISSING AN ENTRY FOR LOCATION %1
If a location entry exists for a horizontal partition, it must be specified for all partitions.
- (ACC20224) WHERE CLAUSE EXCLUDES ALL AVAILABLE PARTITIONS
The WHERE/IF clause in the current request does not match any of the WHERE clauses specified in the Access File. No partitions can be accessed.
- (ACC20225) UNABLE TO LOAD ACCFILE MODULE
The system is unable to find a required library.
- (ACC20226) DATASET SPECIFIED FOR MASTER %1 DOES NOT EXIST
The dataset specified in the Master either has not been created or cannot be read. Verify the status of the specified physical file and try the request again.

(ACC20227) PARTITION No. %2 FOR DATABASE %1 DOES NOT EXIST
The partition specified in the Access File for the above Master either has not been created or cannot be read. Verify the status of the physical partition.

(ACC20228) LOCATION %3 IN PARTITION No. %2 FOR DATABASE %1 DOES NOT EXIST
The physical file specified for the above LOCATION for the specified partition in the given database either does not exist or cannot be read. Verify the status of the file and retry the request.

(ACC20235) INVALID PHYSICAL FILENAME %2 FOR MASTER %1
Only absolute filenames are accepted by the Access File and the DATASET command.

NF778: Dialogue Manager TRUNCATE Function

The Dialogue Manager TRUNCATE function removes trailing blanks from Dialogue Manager ampers variables and adjusts the length accordingly.

Using the Dialogue Manager TRUNCATE Function

The Dialogue Manager TRUNCATE function has only one argument, the string or variable to be truncated. If you attempt to use the Dialogue Manager TRUNCATE function with more than one argument, the following error message is generated:

```
(FOC03665) Error loading external function 'TRUNCATE'
```

This function can only be used in Dialogue Manager commands that support subroutine calls, such as -SET and -IF commands. It cannot be used in -TYPE or -CRTFORM commands or in arguments passed to stored procedures.

Note: A user-written subroutine of the same name can exist without conflict.

Syntax How to Use the TRUNCATE Function

```
-SET &var2 = TRUNCATE(&var1);
```

where:

&var2

Is the Dialogue Manager variable to which the truncated string is returned. The length of this variable is the length of the original string or variable minus the trailing blanks. However, if the original string consisted of only blanks, a single blank, with a length of one is returned.

&var1

Is a Dialogue Manager variable or a literal string enclosed in single quotation marks. System variables and statistical variables are allowed as well as user-created local and global variables.

Example Using the Dialogue Manager TRUNCATE Function

The following example shows the result of truncating trailing blanks:

```
-SET &LONG = 'ABC   ' ;  
-SET &RESULT = TRUNCATE(&LONG);  
-SET &LL = &LONG.LENGTH;  
-SET &RL = &RESULT.LENGTH;  
-TYPE LONG   = &LONG   LENGTH = &LL  
-TYPE RESULT = &RESULT LENGTH = &RL
```

The output is:

```
LONG   = ABC   LENGTH = 06  
RESULT = ABC LENGTH = 03
```

The following example shows the result of truncating a string of all blanks:

```
-SET &LONG = '       ' ;  
-SET &RESULT = TRUNCATE(&LONG);  
-SET &LL = &LONG.LENGTH;  
-SET &RL = &RESULT.LENGTH;  
-TYPE LONG   = &LONG   LENGTH = &LL  
-TYPE RESULT = &RESULT LENGTH = &RL
```

The output is:

```
LONG   =           LENGTH = 06  
RESULT =           LENGTH = 01
```

The following example uses the TRUNCATE function as an argument for EDIT:

```
-SET &LONG = 'ABC   ' ;  
-SET &RESULT = EDIT(TRUNCATE(&LONG) | 'Z', '9999');  
-SET &LL = &LONG.LENGTH;  
-SET &RL = &RESULT.LENGTH;  
-TYPE LONG   = &LONG   LENGTH = &LL  
-TYPE RESULT = &RESULT LENGTH = &RL
```

The output is:

```
LONG   = ABC   LENGTH = 06  
RESULT = ABCZ LENGTH = 04
```

NF779: FOCUS CRTFORM HTML Translation

When the HTML/TP feature of Web390 generates replacement HTML forms for a 3270 screen, it can dynamically account for fields that may or may not be populated with data during execution. HTML/TP can use this technique with turnaround (T.) fields on CRTFORMs because they are enclosed in @ signs. These @-sign markers enable HTML/TP to recognize them and handle them dynamically on a customized HTML form. In contrast, CRTFORM display (D.) fields are not normally enclosed in @ signs.

A new SET parameter, WEBTAB, can be used to instruct FOCUS to enclose CRTFORM display fields in @ signs.

Note: This setting is only for those MODIFY CRTFORM or Dialogue Manager -CRTFORM applications that will be used in conjunction with the HTML/TP feature of Web390. For information about Web390 and the HTML/TP feature, see the *Web390 for OS/390 and MVS Developer's Guide and Installation Manual*.

Syntax How to Add Markers to CRTFORM Display Fields

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

where:

ON

Adds @ signs around CRTFORM display fields. These markers may cause the fields displayed on the CRTFORM to shift slightly to the right. Use this setting only for MODIFY CRTFORM or Dialogue Manager -CRTFORM applications that will be used in conjunction with the HTML/TP feature of Web390.

OFF

Does not place @ signs around CRTFORM display fields. OFF is the default setting.

Syntax **How to Query the WEBTAB Parameter Setting**

The WEBTAB parameter setting is displayed by ? SET WEBTAB query command.

? SET WEBTAB

NF781: Embedding Text Fields in Headings

You can now embed text fields (FORMAT=TXn) in heading types (heading, footing, subhead, subfoot, and ON TABLE subhead and subfoot) in reports.

Example Embedding a Text Field in a Heading

The following example illustrates how to embed a text field in a heading.

```
TABLE FILE TXTFLD
BY COUNTRY SUBHEAD
"Here is a TX field: <TEXTFLD"
END
```

The heading in the output is:

```
Here is a TX field: The quick brown
                    fox jumps over
                    the lazy dog.
```

The following is the Master File for this example:

```
FILENAME=TXTFLD,SUFFIX=FOC
SEGNAME=ORIGIN,SEGTYPE=S1
FIELDNAME=COUNTRY,FORMAT=A16,$
FIELDNAME=TEXTFLD,FORMAT=TX16,$
```

As is shown in this example, the text field values display on one or more lines. The output is aligned vertically so that the position of the TX field in the initial line is maintained in the following lines.

The number of characters specified in the TX format specification determines the horizontal space occupied by the text field (for example, TX16 means 16 spaces wide).

Reference Usage Notes for Embedding Text Fields in Headings

- You cannot embed TX fields in FML free-text lines.
- HEADING or FOOTING lines can contain multiple embedded TX fields; SUBHEAD or SUBFOOT lines may only contain a single embedded TX field instance.

NF782: Oracle Data Adapter IXSPACE Setting

The new IXSPACE setting for the Oracle Data Adapter enables you to override the default parameters for the Oracle index space implicitly created by the CREATE FILE and HOLD FORMAT SQLORA commands.

Specifying Oracle Index Space Parameters

You can specify up to 94 bytes of index space parameters for the SQL CREATE INDEX statement that the data adapter issues when it creates an Oracle table as a result of a CREATE FILE or HOLD FORMAT SQLORA command for a table that has an index defined.

Syntax How to Specify Oracle Index Space Parameters

You can issue the following SET command at the command prompt, in a stored procedure, or in any supported profile

```
SQL [SQLORA] SET IXSPACE [index_spec]
```

where:

index_spec

Consists of up to 94 bytes of valid Oracle index space parameters. To reset the index space parameters to their default values, issue the SET IXSPACE command with no parameters. If the index specification requires more than one line, use the long form of the SQL SET command:

```
SQL [SQLORA]  
SET IXSPACE  
index_spec_line1  
index_spec_line2  
END
```

SQLORA

Indicates the Oracle RDBMS. Omit if you issued the SET SQLENGINE command for Oracle.

Examples of index specification parameters you may want to specify are the tablespace clause, global_index clause, local_index clause, and parallel clause. The data adapter does not verify the validity of the parameters. See the *Oracle SQL Reference Manual* for syntax descriptions.

Example Specifying a Table Space for an Oracle Index

When you issue the CREATE FILE or HOLD FORMAT SQLORA command, the DBSPACE setting determines which table space will be used to contain the resulting Oracle table. In prior releases, the index was always placed in the default table space specified at installation.

You can now specify a table space for the index using the SQL SET IXSPACE command. The following example places the index in the tablespace ORATS1:

```
SQL SQLORA SET IXSPACE TABLESPACE ORATS1
```

As a result of setting this value, any CREATE FILE or HOLD FORMAT SQLORA command will generate an SQL CREATE INDEX statement containing the TABLESPACE clause, if an index is defined for the table. You can see the SQL CREATE INDEX syntax in the FSTRACE4 output.

The data adapter does not verify the validity of the parameters. If they are not valid, the following errors are returned at runtime:

```
(FOC1400)      SQLCODE IS 959 (HEX: 000003BF)
                : ORA-00959: tablespace 'tablename' does not exist

(FOC1414)      EXECUTE IMMEDIATE ERROR.
```

When the SQL SQLORA SET IXSPACE TABLESPACE *tablespacename* command has been issued with a valid table space name, the following line is added to the SET SQLORA? query command output:

```
(FOCxxxx) DEFAULT IXSPACE IS           - : tablespacename
```

If no table space has been set, this line is not displayed.

Example Resetting the IXSPACE Value

The following command resets the IXSPACE parameters to their installation defaults:

```
SQL SQLORA SET IXSPACE
```

NF785: The Adabas Write Data Adapter for FOCUS

You can now use FOCUS to maintain Adabas data sources. This bulletin identifies aspects of the FOCUS file maintenance facilities, MODIFY and Maintain, that are unique for the Adabas environment. MODIFY and Maintain requests read, add, update, and delete records. You can modify single data sources, sets of data sources defined in a multi-segment Master File, or unrelated sets of data sources.

Maintain provides a graphical user interface and event-driven processing. In a Maintain procedure, temporary storage areas called stacks collect data, transaction values, and temporary field values. You can use the Maintain Window Painter facility to design Winforms—windows that display stack values, collect transaction values, and invoke triggers. A trigger implements event-driven processing by associating an action (such as performing a specific case in the Maintain procedure) with an event (such as pressing a particular PF key). Maintain also provides set-based processing through enhanced NEXT, UPDATE, DELETE, and INCLUDE commands.

This bulletin describes differences between the MODIFY and Maintain facilities when these differences affect data adapter processing.

The FOCUS for S/390 documentation contains a detailed discussion of file maintenance with the FOCUS MODIFY and Maintain facilities. Read this documentation carefully before developing MODIFY or Maintain procedures to use with Adabas data sources.

Note: You can maintain up to 64 data sources in a single MODIFY or Maintain procedure. The limit for a MODIFY COMBINE or a Maintain procedure is 16 Master Files; however, each Master File can describe more than one data source, for a total of 64 segments per procedure minus one for the virtual root segment created by the COMBINE command. In addition, a Maintain procedure can call other Maintain procedures that reference additional data sources.

Prerequisites for running MODIFY and Maintain requests include:

- The Write data adapter. The Write component of the data adapter must be installed and operational.
- Proper authorization to perform maintenance operations.
- Existing data sources to modify. If you intend to load new data sources with MODIFY or Maintain, you must generate them first using Adabas utilities.

Activating the Adabas Write Data Adapter

Before you can use any commands that write to an Adabas data source, you must make the following edits to the Master and Access Files:

- In the Master File:
 - All segments must have SEGTYPE = S0. (For the Read Data Adapter SEGTYPE=S is sufficient.)
 - Fields with ACTUAL format Z (zoned) must have a numeric USAGE format (P or I). Format A is supported only for reading an Adabas data source.

Tip: For best performance, you can change ACTUAL format Z to ACTUAL format P in the Master File. In this case, you must also change the ALIAS attribute to contain the length and data type. If the ALIAS was *ff*, and the field length is *lll*, the ALIAS attribute should be coded as:

```
ALIAS = 'ff,lll,P'
```

For example, consider the following field declaration:

```
FIELD = LEAVE_DUE ,ALIAS = AU ,USAGE = P2 ,ACTUAL = Z2 , $
```

You would edit this declaration as follows:

```
FIELD = LEAVE_DUE ,ALIAS = 'AU,2,P' ,USAGE = P2 ,ACTUAL = P2 , $
```

- If two or more fields in the Master File are synonyms (they refer to the same field in the data source and, therefore, have the same ALIAS attribute), only the first field encountered in the Master File can be used in INSERT and UPDATE commands. If a synonym other than the first is used in an INSERT command, it will be ignored. If one is used in an UPDATE command, processing will be terminated with the following error message:

(FOC4565) IGNORED ATTEMPT TO CHANGE NONUPDATABLE FIELD

- In the Access File:
 - If a segment has the attribute ACCESS=ADBS you can define a unique key by adding the following attribute:

UNQKEYNAME=*name*

where:

name

Is the name of the elementary or group field to be used as the unique key.

The UNQKEYNAME attribute does not necessarily define the key described in the Adabas FDT (option UQ). The data adapter uses it to decide which rules to apply in a NEXT, INCLUDE, DELETE, or UPDATE command. If the UNQKEYNAME attribute does not correspond to the key described with option UQ in the Adabas FDT, Adabas and the data adapter may not agree on whether a segment instance is unique. This can affect the results of INCLUDE commands, as described in [Effect of UNQKEYNAME on INCLUDE Actions for Segments With ACCESS=ADBS](#).

If this attribute is not present, the data adapter uses the rules for modifying a segment with a non-unique key. If it is present, the data adapter uses the rules for modifying a segment with a unique key. Subsequent sections describe these rules.

Note: For modifying Adabas data sources with Maintain, the UNQKEYNAME attribute in the Access File is required. Otherwise you can use only commands that do not change the data source (such as TYPE) and can only issue requests against single-segment data sources.

- We recommend the use of CALLTYPE=FINN instead of CALLTYPE=RL.

Reference AUTOADBS Considerations

You can use AUTOADBS to create a read version of the Master File. To convert it to a Write version, change the SEGTYPE value in all segments to S0.

Limitations on Options Described for the Adabas Data Adapter

When using the Write data adapter, the data adapter automatically sets the values of the following two options:

- ADABAS OPEN is set to YES.
- FETCH is set to OFF for all segments.

Fields That Cannot be Updated

For both MODIFY and Maintain, certain types of fields listed in the Master File cannot be updated. In some cases, the update commands are ignored, and in some cases they generate errors. The following sections contain a detailed explanation.

Reference Using Synonyms

In a Master File, two or more field declarations can refer to the same Adabas field. Each duplicate field declaration after the first is called a synonym. Synonyms can be used in commands that print, but cannot be used in commands that change the data source. The following actions occur as a result of using synonyms in INCLUDE/UPDATE commands:

- Synonym fields used in INCLUDE commands are ignored.
- Synonym fields used in UPDATE commands cause processing to terminate and generate the following error message:

(FOC4565) IGNORED ATTEMPT TO CHANGE NONUPDATABLE FIELD

An UPDATE command was used against a nonupdatable field,
i.e. - field described in AFD as SUB/SUPERDESCRIPTOR or part
of it;
- field that refers in MFD to the same DB field (ALIAS) as another
field (synonym case).

Reference Fields That Cannot Be Updated in MODIFY

In MODIFY, update commands for the following types of fields are ignored:

- Counter fields (ALIAS=xxC).
- ORDER fields (ALIAS=ORDER).
- Group fields (UPDATE ignored).

In MODIFY, update commands for the following types of fields generate an error:

- Synonyms (see [Using Synonyms](#)).
- Fields created from sub- or superdescriptors (as defined in the Access File).

Reference Fields That Cannot Be Updated in Maintain

Maintain generates an error in response to update commands for the following types of fields:

- Counter fields (ALIAS=*xx*C).
- ORDER fields (ALIAS=ORDER).
- Unique key fields (fields specified by the UNQKEYNAME attribute in the Access File).
- Synonyms (see [Using Synonyms](#)).

Checking Adabas Return Codes and FOCUS Error Message Numbers

The Dialogue Manager status return variable, &RETCODE, indicates the status of FOCUS requests. You can use it to test Adabas return codes. The &RETCODE variable contains the last return code resulting from an executed report request or MODIFY request.

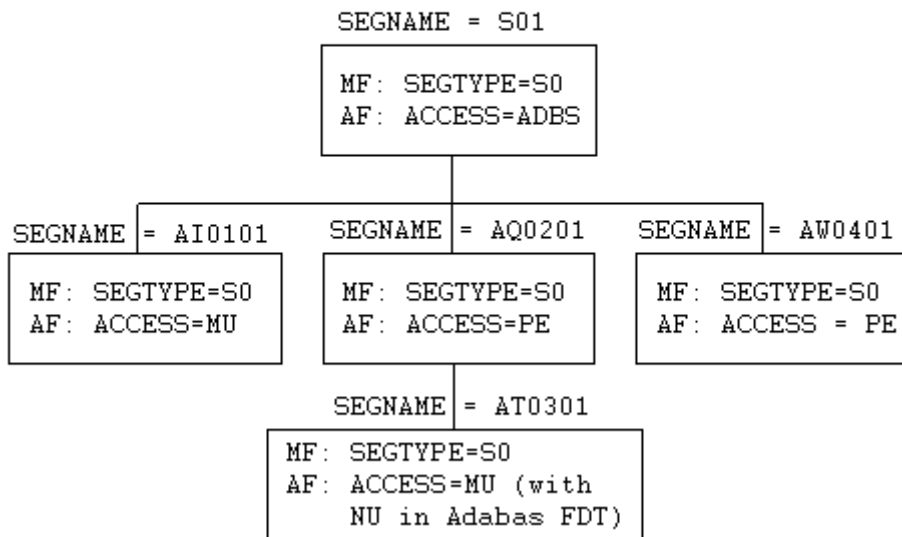
In a Dialogue Manager request, you can use a -IF statement to test the &RETCODE value against a specified Adabas return code. Then, you can take corrective actions based on the result of the -IF test. An Adabas return code of zero (0) indicates a successful execution, a non-zero return code indicates an error.

Another useful Dialogue Manager variable, &FOCERRNUM, stores the last FOCUS or data adapter (not Adabas) error number generated by the execution of a FOCEXEC. See your FOCUS for S/390 documentation for information about &FOCERRNUM and other statistical variables.

Adabas Write Examples

The examples in this document access the EMPLOYEES-WRITE data source.

Adabas Structure of EMPWRITE View
of EMPLOYEES-WRITE Data Source



Example Sample Adabas Write Master File

This Master File provides write access to the Adabas EMPLOYEES-WRITE data source:

```
FILENAME=EMPWRITE,SUFFIX=ADBSIN,$
SEGNAME=S01,SEGTYPE=S0,$
FIELD= EMPLOYEE_ID,ALIAS= AA,A8,A8,INDEX=I,$
GROUP= FULL_NAME,ALIAS= AB,A60,A60,$
FIELD=FIRST_NAME,ALIAS= AC,A20,A20,$
FIELD=LAST_NAME,ALIAS= AE,A20,A20,INDEX=I,$
FIELD=MIDDLE_NAME,ALIAS= AD,A20,A20,$
FIELD=ADDRESS_LINE_CNT,ALIAS= AIC,I4,I1,$
FIELD=CITY,ALIAS= AJ,A20,A20,INDEX=I,$
FIELD=ZIP_CODE,ALIAS= AK,A10,A10,$
FIELD=POST_CODE,ALIAS= AK,A10,A10,$
FIELD=COUNTRY,ALIAS= AL,A3,A3,$
FIELD= DEPT,ALIAS= AO,A6,A6,INDEX=I,$
FIELD= INCOME_CNT,ALIAS= AQC,I4,I1,$
GROUP= LEAVE_DATA,ALIAS= A3,A16,A4,$
FIELD=LEAVE_DUE,ALIAS= AU,P2,Z2,$
FIELD=LEAVE_TAKEN,ALIAS= AV,P2,Z2,$
FIELD= LEAVE_BOOKED_CNT,ALIAS= AWC,I4,I1,$
FIELD= DEPARTMENT,ALIAS= S1,A4,A4,INDEX=I,$
GROUP= DEPT_PERSON,ALIAS= S2,A26,A26,INDEX=I,$
FIELD=DEPT_S03,ALIAS= AO,A6,A6,INDEX=I,$
FIELD=NAME_S03,ALIAS= AE,A20,A20,INDEX=I,$
SEGNAME=AI0101,SEGTYPE=S0,PARENT=S01,OCCURS=AIC,$ MAX= 8
FIELD= ADDRESS_LINE,ALIAS= AI,A20,A20,$
FIELD= AI0101_OCC,ALIAS= ORDER,I4,I1,$
SEGNAME=AQ0201,SEGTYPE=S0,PARENT=S01,OCCURS=AQC,$ MAX= 40
$PEMU = INCOME,ALIAS= AQ,A19,A13,$
FIELD=CURR_CODE,ALIAS= AR,A3,A3,$
FIELD=SALARY,ALIAS= AS,P9,P5,$
FIELD=BONUS_CNT,ALIAS= ATC,I4,I1,$
FIELD= AQ0201_OCC,ALIAS= ORDER,I4,I1,$
```

```
SEGNAME=AT0301 ,SEGTYPE=S0 ,PARENT=AQ0201 ,OCCURS=ATC,$ MAX= 12
  FIELD= BONUS ,ALIAS= AT ,P9 ,P5 ,,$
  FIELD= AT0301_OCC ,ALIAS= ORDER ,I4 ,I1 ,,$
```

```
SEGNAME=AW0401 ,SEGTYPE=S0 ,PARENT=S01 ,OCCURS=AWC,$ MAX= 20
  GROUP= LEAVE_BOOKED ,ALIAS= AW ,A16 ,A12 ,,$
  FIELD=LEAVE_START ,ALIAS= AX ,P6 ,Z6 ,,$
  FIELD=LEAVE_END ,ALIAS= AY ,P6 ,Z6 ,,$
  FIELD= AW0401_OCC ,ALIAS= ORDER ,I4 ,I1 ,,$
```

Example Sample Adabas Write Access File

The following Access File corresponds to the Master File illustrated in [Sample Adabas Write Master File](#):

```
RELEASE=6.2 , OPEN=YES,$

$ ADABAS FILE = EMPLOYEES_WRITE DICTONARY =
SEGNAM=S01 ,ACCESS=ADBS ,FILENO=042 ,DBNO=1 ,CALLTYPE=FINND ,
UNQKEYNAME=EMPLOYEE_ID ,,$
FIELD= DEPARTMENT ,TYPE=NOP ,,$
FIELD= DEPT_PERSON ,TYPE=SPR ,,$
FIELD=DEPT_S03 ,TYPE=DSC ,NU=NO ,,$
FIELD=NAME_S03 ,TYPE=DSC ,NU=NO ,,$
SEGNAM=AI0101 ,ACCESS=MU ,FILENO=042 ,DBNO=1 , $ ADDRESS_LINE
SEGNAM=AQ0201 ,ACCESS=PE ,FILENO=042 ,DBNO=1 , $ INCOME
SEGNAM=AT0301 ,ACCESS=MU ,FILENO=042 ,DBNO=1 , $ BONUS
SEGNAM=AW0401 ,ACCESS=PE ,FILENO=042 ,DBNO=1 , $ LEAVE_BOOKED
```

Types of Transaction Processing

You can process incoming transactions by comparing (or matching on) the following types of fields:

- For segments with a unique key:
 - A unique key field or superset of the key (unique key plus any number of non-unique key fields or non-key fields).
 - A non-unique key field or a non-key field that is used as a non-unique key field, as described in [Descriptor Considerations](#).
- For segments with a non-unique key:
 - Any field when issuing NEXT or INCLUDE commands in a subrequest (after at least one initial MATCH or NEXT command). Any field can be used as a unique key for these commands.

In MODIFY, a MATCH on a non-unique key in a segment with a unique key may retrieve more than one record instance. MATCH returns only the first instance of this answer set; subsequent sections demonstrate how to use NEXT to retrieve the remaining instances.

In Maintain, MATCH always matches on the unique key and retrieves at most one record instance. To match on a non-unique key in Maintain, you use the NEXT command without a prior MATCH. The Maintain implementation of the NEXT command fetches the entire answer set returned by Adabas directly into a stack. It also includes three optional phrases:

- The FOR phrase determines how many records to retrieve.
- The WHERE phrase defines retrieval criteria.
- The INTO phrase names a stack to receive the returned records.

See the FOCUS for S/390 documentation for complete syntax.

Descriptor Considerations

Descriptors (or inverted lists) enhance the performance of data maintenance routines, especially when created on the data source's unique key. Without a descriptor, Adabas must read the entire data source to locate particular record instances; with a descriptor, Adabas can access record instances directly when given search values for the descriptor fields. A data source can have several associated descriptors. Descriptors created for performance reasons can be unique or non-unique. To use FOCUS referential integrity, create descriptors on lower level segments in the Master File.

If all of the following conditions are true, non-descriptor (non-key) fields can be used as non-unique descriptor fields:

- The Access File defaults to or specifies `CALLTYPE=FOUND`.
- The command `SET NDFIND=NO` was not issued (by default, `NDFIND=YES`).
- The Adabas session parameter `NONDES` is set to `YES` (the default).

You can define a unique key in a data source. When a key is unique, the concatenated values of the key fields in one record cannot be duplicated in any other record. A unique descriptor is generally defined on a unique key. Once you create it, Adabas automatically prevents the insertion of duplicate key values. Any attempt to insert a duplicate record instance generates an error message.

An example of a unique descriptor on a unique key is the employee ID in the sample EMPWRITE data source. Since no two employees can have the same employee number, the value in the EMPLOYEE_ID field makes each record unique:

EMPLOYEE_ID	LAST_NAME	FIRST_NAME
-----	-----	-----
50004300	GUERIN	MICHELE
50004600	VERDIE	BERNARD
50004900	CAUDAL	ALBERT
50005500	BRAUN	ALEXANDRE
50005800	GUENTER	SIMONE

You cannot add another record with EMPLOYEE_ID 50004300 to this data source.

Any segment below the root always has a unique key—the ORDER field. You can also use another field as either a non-unique key field (for example, SALARY) or as a unique key, if you are sure that none of the values are duplicated (for example, LEAVE_START).

Modifying Data

With the FOCUS MODIFY and Maintain facilities, you can add new records to a data source, update field values for specific records, or delete specific records.

The data adapter processes a MODIFY or Maintain transaction with the following steps:

1. FOCUS reads the transaction for incoming data values.
2. The data adapter generates the appropriate calls to Adabas based on the MATCH or NEXT criteria.
3. Adabas either returns an answer set consisting of one or more records that satisfy the request, or determines that the record does not exist.

4. After Adabas returns the answer set and/or return code, the data adapter either
 - Performs the update operation (UPDATE or DELETE) on the returned answer set. With MODIFY, the data adapter processes one record at a time; with Maintain, it can either process one record at a time or a set of records.
 - Creates the new record (INCLUDE). In Maintain, it may create multiple records.
5. Adabas changes the database appropriately.

In MODIFY, you must use the NEXT command to process a multi-record answer set one record at a time. Each NEXT command puts you physically at the next logical record in the answer set created by the most recent MATCH command for this segment. In Maintain, one NEXT command can process a multi-record answer set without a prior MATCH.

The MATCH Command

In response to a MATCH command, the data adapter selects the first record in the data source that meets the MATCH criteria.

The MATCH command compares incoming data with one or more field values and then performs actions that depend on whether or not a record with matching field values exists in the data source.

The syntax of the MATCH command in MODIFY is

```
MATCH field1 [ field2...fieldn ]  
  ON MATCH action_1  
  ON NOMATCH action_2
```

where:

fieldn

Are fields in any segment of the Master File. FOCUS compares incoming data values against existing field values. The fields can be any combination of key and/or non-key fields. Specify complete fieldnames; MATCH does not support truncated names.

action_1

Is the operation to perform when a record's values match the incoming data values.

action_2

Is the operation to perform when a record's existing values do not match the incoming data values.

The FOCUS for S/390 documentation discusses these actions in detail.

MATCH processing for multi-segment Master Files is the same as for a multi-segment FOCUS database.

Acceptable actions for MATCH commands fall into eight groups. They are operations that:

- Include, change, or delete records.
- Control MATCH processing, such as rejecting the current transaction.
- Read incoming data fields.
- Perform computations and validations, or type messages to the terminal.
- Control case logic.
- Control multiple-record processing.
- Activate and deactivate fields in MODIFY.
- Permanently store data in the Adabas data source.

Data Adapter MATCH Behavior

In MODIFY requests, there are two major differences in the way MATCH commands function for the data adapter and for native FOCUS:

- With the data adapter, you can change the value of a data source's unique key (subject to Adabas limitations) using the UPDATE command. When modifying a FOCUS data source, you cannot change key field values.
- You can MATCH on *any* field or combination of fields in the record. However, if the full unique key is not included in the MATCH criteria, the data adapter may retrieve more than one record as a result of the MATCH. For example, if the unique key is EMPLOYEE_ID and the incoming value for MATCH LAST_NAME is SMITH, the answer set contains all records with last name SMITH.

Note: In Maintain, MATCH functions identically for the data adapter and for native FOCUS.

Example Using the MODIFY MATCH Command

Consider a MODIFY request that maintains the EMPWRITE data source. It prompts for an employee ID and for a new number of leave days; then it processes the incoming data. The annotated request contains the following MATCH commands:

```
MODIFY FILE EMPWRITE
PROMPT EMPLOYEE_ID LEAVE_DUE
1. MATCH EMPLOYEE_ID
2. ON MATCH UPDATE LEAVE_DUE
3. ON NOMATCH REJECT
DATA
```

The incoming transaction contains the following values:

```
EMPLOYEE_ID = 12345678  
LEAVE_DUE = 20
```

The request processes as follows:

1. The MATCH command compares the value of the incoming EMPLOYEE_ID, 12345678, to the EMPLOYEE_ID values in the records of the EMPWRITE data source. Since EMPLOYEE_ID is the unique key of this data source, Adabas can return at most one record as a result of this MATCH.
2. If a record exists for EMPLOYEE_ID 12345678, the MATCH command updates the LEAVE_DUE value of that record with the incoming value 20.
3. If no record exists for EMPLOYEE_ID 12345678, the MATCH command rejects the transaction.

In Maintain, you do not have to include an ON NOMATCH command in order to reject a transaction; Maintain automatically rejects a transaction that does not satisfy the MATCH criteria.

The NEXT Command

In MODIFY, the NEXT command provides a flexible means of processing multi-record answer sets by moving the current position in the answer set from one record to the next.

The syntax is

```
NEXT field  
    ON NEXT action_1  
    ON NONEXT action_2
```

where:

field

Is any field in the current segment. This field does not affect subsequent actions.

action_1

Is the operation to perform when there is a subsequent record in the answer set. May be any of the acceptable actions listed for MATCH in [The MATCH Command](#).

action_2

Is the operation to perform when no more records exist in the answer set. The CALLTYPE parameter in the Access File controls the sort order for NEXT. It determines whether to retrieve records in physical order (CALLTYPE=FINDD) or sorted by the unique key (CALLTYPE=RL).

Your choice of MATCH and NEXT command combinations determines the contents of the answer set. Subsequent sections explain these choices in more detail:

- NEXT command without a MATCH command. The data adapter requests the retrieval of all records in the data source in physical order (CALLTYPE=FINDD) or sorted by the unique key (CALLTYPE=RL).
- MATCH on the unique key. The MATCH returns the single record that is the starting point for any subsequent NEXT commands, which retrieve the remaining records in the answer set in physical order (CALLTYPE=FINDD) or sorted by the unique key (CALLTYPE=RL).
- MATCH on a non-unique key field. Adabas returns a multi-record answer set in which each record satisfies the MATCH criteria. The data adapter requests the retrieval of these records in physical order.

You can also use NEXT commands with multi-segment structures (FOCUS views) to modify or display data in either case logic or non-case logic requests. If your MATCH or NEXT specifies a record from a parent segment in a multi-segment structure, that record becomes the current position in the parent segment.

A subsequent NEXT on a field in a descendant of that segment retrieves the first descendant record in the related segment. In MODIFY:

- Without case logic, you can retrieve all parent records in the segment and only the first descendant record of any specified related segment.
- With case logic, you can retrieve all records for each segment defined in a multi-segment Master File. To do so, first MATCH on the parent. Then, in another case, use NEXT to loop through the related segments (at the lowest level) until there are no more related instances. On NONEXT, return to the parent case for the next parent instance.

You can trace case logic with the FOCUS trace facility. To invoke the trace facility, include the TRACE command on a separate line after the MODIFY FILE command. You can also use the data adapter trace facilities, described in Technical Memo 7966, *Adabas Interface: Using Traces*.

The following sections illustrate different combinations of MATCH and NEXT command with annotated examples. The MODIFY requests have been kept simple for purposes of illustration; you can create more sophisticated procedures.

Note: In Maintain:

- The syntax of the NEXT command includes optional FOR and WHERE phrases that control the number of records retrieved into a stack. As in MODIFY, the CALLTYPE attribute in the Access File determines whether NEXT returns records in physical order or unique key order.
- NEXT always starts its retrieval at the current database position; it will not retrieve a record it has already passed in its retrieval path unless you use the REPOSITION command to reset the current position.

Also as in MODIFY, once you MATCH on a parent segment, a subsequent NEXT on a child segment retrieves descendant records within the parent established by the MATCH. However, one NEXT command can retrieve all such child instances, without case logic.

- The UPDATE, DELETE, and INCLUDE commands also incorporate the optional FOR phrase to process multiple records from a stack.

For complete details, see the *Maintaining Databases* manual.

NEXT Processing Without MATCH

If you use a NEXT command without a previous MATCH command in a MODIFY request, Adabas returns an answer set consisting of all records in the data source in physical order (CALLTYPE= FIND) or sorted by the unique key (CALLTYPE= RL). Use the ON NEXT command to view each record in the order determined by the CALLTYPE attribute. In a Maintain request, the FOR and WHERE phrases in the NEXT command determine the number of records retrieved, in the order determined by the CALLTYPE attribute.

Example Using NEXT Without MATCH in MODIFY

In this MODIFY example, the NEXT command retrieves each record in physical order because the Access File contains the attribute CALLTYPE= FIND:

```
MODIFY FILE EMPWRITE
NEXT EMPLOYEE_ID
  ON NEXT TYPE "EMPLOYEE ID: <D.EMPLOYEE_ID LAST NAME: <D.LAST_NAME "
  ON NONEXT GOTO EXIT
DATA
END
```

The TYPE commands display the following on the screen:

```
EMPLOYEE ID: 50005800 LAST NAME: GUENTER
EMPLOYEE ID: 50005500 LAST NAME: BRAUN
EMPLOYEE ID: 50004900 LAST NAME: CAUDAL
EMPLOYEE ID: 50004600 LAST NAME: VERDIE
EMPLOYEE ID: 50004300 LAST NAME: GUERIN
EMPLOYEE ID: 50004200 LAST NAME: VAUZELLE
EMPLOYEE ID: 50004100 LAST NAME: CHAPUIS
EMPLOYEE ID: 50004000 LAST NAME: MONTASSIER
EMPLOYEE ID: 50003800 LAST NAME: JOUSSELIN
EMPLOYEE ID: 50006900 LAST NAME: BAILLET
.
.
.
```

If the Access File contained the attribute CALLTYPE=RL, the records would be retrieved in order of employee ID number.

Example Using NEXT in Maintain

The following Maintain procedure retrieves the same answer set into a stack named INSTACK and displays the retrieved values on a Winform named WIN1 (consult the *Maintaining Databases* manual for instructions on creating Winforms):

```
MAINTAIN FILE EMPWRITE
INFER EMPLOYEE_ID LAST_NAME INTO INSTACK
FOR ALL NEXT EMPLOYEE_ID INTO INSTACK
WINFORM SHOW WIN1
END
```


The following Winform displays as a result of this procedure:

NEXT without prior MATCH

EMPLOYEE_ID	LAST_NAME
50005800	GUENTER
50005500	BRAUN
50004900	CAUDAL
50004600	VERDIE
50004300	GUERIN
50004200	VAUZELLE
50004100	CHAPUIS
50004000	MONTASSIER
50003800	JOUSSELIN

Exit

NEXT Processing After MATCH on a Unique Key

In MODIFY, NEXT processing is identical for either MATCH on a full unique key or MATCH on a superset (full unique key plus a non-key field).

When the initial MATCH is successful, Adabas retrieves one record. This establishes the logical position in the data source. The subsequent NEXT command causes Adabas to retrieve all records following the matched record in physical order (CALLTYPE= FIND) or key sequence (CALLTYPE=RL).

Example Using NEXT After MATCH on a Full Unique Key in MODIFY

The following is an example of NEXT processing after a MATCH on a full unique key, the EMPLOYEE_ID field:

```
MODIFY FILE EMPWRITE
CRTFORM LINE 1
" PLEASE ENTER VALID EMPLOYEE ID </1"
1. " EMP: <EMPLOYEE_ID  "
2. MATCH EMPLOYEE_ID
   ON NOMATCH REJECT
3. ON MATCH GOTO GETREST
   CASE GETREST
4. NEXT EMPLOYEE_ID
   ON NEXT CRTFORM LINE 10
   " EMPLOYEE_ID: <D.EMPLOYEE_ID   LAST_NAME: <D.LAST_NAME  "
   ON NEXT GOTO GETREST
5. ON NONEXT GOTO EXIT
   ENDCASE
   DATA
   END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID for the search, 20009000.
2. The MATCH command causes Adabas to search the data source for the entered value. If no such record exists, the transaction is rejected.
3. If the specified value matches a value in the EMPLOYEE_ID field of the data source, the procedure branches to the GETREST case; it contains the NEXT command.

4. The NEXT command retrieves the next record based on physical order, if the Access File contains the attribute CALLTYPE=FINN, or based on the sequence of EMPLOYEE_ID, if the Access File specifies CALLTYPE=RL. If such a record exists, the procedure displays the values of the EMPLOYEE_ID and LAST_NAME fields. It continues to display each record in the order determined by the CALLTYPE attribute of the key field, EMPLOYEE_ID.
5. If there are no more records, the procedure ends.

The output after executing this MODIFY procedure with CALLTYPE=FINN is:

PLEASE ENTER VALID EMPLOYEE ID (line 1)

EMP: 20009000 (line 3)

EMPLOYEE_ID: 50005800 LAST_NAME: GUENTER (line 10)

EMPLOYEE_ID: 50005500 LAST_NAME: BRAUN (line 10)

EMPLOYEE_ID: 50004900 LAST_NAME: CAUDAL (line 10)

EMPLOYEE_ID: 50004600 LAST_NAME: VERDIE (line 10)

EMPLOYEE_ID: 50004300 LAST_NAME: GUERIN (line 10)

Because of the NEXT command, all employees whose records are physically after 20009000 display one at a time on the screen.

The output after executing this MODIFY procedure with CALLTYPE=RL is:

```
PLEASE ENTER VALID EMPLOYEE ID                                (line 1)

EMP: 20009000                                                (line 3)

EMPLOYEE_ID: 20009100   LAST_NAME:  JENSON                    (line 10)
EMPLOYEE_ID: 20009200   LAST_NAME:  MEYER                    (line 10)
EMPLOYEE_ID: 20009300   LAST_NAME:  SMITH                    (line 10)
EMPLOYEE_ID: 20009400   LAST_NAME:  OLLE                     (line 10)
EMPLOYEE_ID: 20009500   LAST_NAME:  RAY                      (line 10)
```

Notice that the employee ids after 20009000 are retrieved in key sequence.

Example Using NEXT on a Full Unique Key in Maintain

The following Maintain procedure retrieves the same answer set into a stack named EMPSTACK. Assume that when Maintain displays the Winform called WIN1, the user enters the transaction value, 20009000, into a stack named TRANS and clicks the NextRecs button to invoke the NEXTRECS case:

```
MAINTAIN FILE EMPWRITE
INFER EMPLOYEE_ID LAST_NAME INTO EMPSTACK
WINFORM SHOW WIN1
CASE NEXTRECS
  FOR ALL NEXT EMPLOYEE_ID INTO EMPSTACK WHERE EMPLOYEE_ID GT
  TRANS.EMPLOYEE_ID
ENDCASE
END
```

The following Winform displays when 20009000 is entered in the EMPLOYEE_ID field with CALLTYPE= FIND:

Enter an employee id: 20009000

EMPLOYEE_ID	LAST_NAME
50005800	GUENTER
50005500	BRAUN
50004900	CAUDAL
50004600	VERDIE
50004300	GUERIN
50004200	VAUZELLE
50004100	CHAPUIS

NextRecs

Exit

NEXT Processing After MATCH on a Non-Unique Key

In a MODIFY request processed by the data adapter, you do not have to MATCH on the full set of unique key fields. You can match on a non-key field or non-unique key. (Maintain always matches on the full unique key, regardless of which fields you specify in the MATCH command.)

When you MATCH on a non-unique key, multiple records may satisfy the MATCH condition. The MATCH operation retrieves the first record of the answer set, and the NEXT command makes the remaining records in the answer set available to the program in physical order. In this case, the order of retrieval for NEXT is always the physical sequence. MATCH on a non-key field should always be processed using CALLTYPE= FIND to prevent the data adapter from issuing an inefficient Read Physical call. For more information see the *FOCUS for IBM Mainframe ADABAS Interface User's Manual and Installation Guide*.

Example Using MATCH on a Non-Unique Key in MODIFY

This annotated procedure is the same procedure described in [Using NEXT After MATCH on a Full Unique Key in MODIFY](#), altered to MATCH on the non-unique key field LAST_NAME. The NEXT operation retrieves the subsequent records from the answer set in physical sequence:

```
MODIFY FILE EMPWRITE
CRTFORM LINE 1
" PLEASE ENTER A LAST NAME </1 "
1. " LAST_NAME: <LAST_NAME </1"
2. MATCH LAST_NAME
   ON NOMATCH REJECT
3. ON MATCH CRTFORM LINE 5
   " EMPLOYEE_ID: <D.EMPLOYEE_ID LAST_NAME: <D.LAST_NAME "
4. ON MATCH GOTO GETSAME
   CASE GETSAME
5. NEXT LAST_NAME
   ON NEXT CRTFORM LINE 10
   " EMPLOYEE_ID: <D.EMPLOYEE_ID LAST_NAME: <D.LAST_NAME "
   ON NEXT GOTO GETSAME
6. ON NONEXT GOTO EXIT
   ENDCASE
   DATA
   END
```

The MODIFY procedure processes as follows:

1. The user enters the last name (LAST_NAME) for the search, SMITH.
2. The MATCH command causes Adabas to search the data source for all records with the value SMITH and return them in physical order. If the value SMITH does not exist, the transaction is rejected.
3. If the incoming value matches a value in the data source, the procedure displays the employee ID and last name. (This is the first record of the answer set.)
4. After displaying the record, the procedure goes to the GETSAME case; it uses NEXT to loop through the remaining records in the answer set.
5. Instead of retrieving the next logical record with a higher key value as in the previous example, the procedure retrieves the next record in the answer set (all records in the answer set have the last name SMITH). If any exist, they display on the screen in physical order.
6. When no more records exist with the value SMITH, the procedure ends.

The output from this MODIFY procedure follows:

```
PLEASE ENTER A LAST NAME
```

```
LAST_NAME  smith
```

```
EMPLOYEE_ID  40000311  LAST_NAME:  SMITH
```

```
EMPLOYEE_ID:  20009300  LAST_NAME:  SMITH
```

```
EMPLOYEE_ID:  20014100  LAST_NAME:  SMITH
```

```
.
```

```
.
```

```
.
```

```
EMPLOYEE_ID:  30038013  LAST_NAME:  SMITH
```

A line displays on the screen for each employee with the last name SMITH. Employee ID 40000311 is the result of the MATCH operation; employee ID 20009300 is the result of the NEXT operation, employee ID 30038013 is the result of the last NEXT operation. Notice that the records are retrieved in physical sequence, and order is not dependent on the CALLTYPE attribute.

Example Using NEXT on a Non-Unique Key Field in Maintain

The following Maintain procedure retrieves the entire answer set into a stack named EMPSTACK. Assume that when Maintain displays the Winform named WINA, the user enters the transaction value (SMITH) into the first record of a stack named TRANS and clicks the NextRecs button to invoke the NEXTRECS case:

```
MAINTAIN FILE EMPWRITE
INFER EMPLOYEE_ID LAST_NAME INTO EMPSTACK
WINFORM SHOW WINA
CASE NEXTRECS
  FOR ALL NEXT EMPLOYEE_ID INTO EMPSTACK WHERE LAST_NAME EQ
  TRANS.LAST_NAME
ENDCASE
END
```


The following Winform displays when the NexRecs button is pressed with SMITH entered in the Last_Name field:

Last_Name : SMITH

EMPLOYEE_ID	LAST_NAME
40000311	SMITH
20009300	SMITH
20014100	SMITH
20015400	SMITH
20018800	SMITH
20023600	SMITH

NextRecs

Exit

INCLUDE, UPDATE, and DELETE Processing

While MATCH and NEXT operations in MODIFY can operate on unique key or non-unique key fields and return single or multi-record answer sets, the MODIFY commands INCLUDE, UPDATE, and DELETE must always identify the target records by their unique key. Therefore, in MODIFY, each update operation affects at most one record. (In Maintain, the FOR phrase in the update command determines the number of records affected.)

Reference Rules for Inserting Records Into an Adabas Data Source

- For a segment with a unique key:

The unique key field value is used to insert the target segment. If any additional MATCH criteria are supplied for a segment in the path, they will be used to qualify that path segment.

An attempt to use MATCH on a non-unique key before an INCLUDE command generates the following error message:

```
(FOC4563)INCORRECT INCLUDE REQUEST FOR NON UNIQUE KEY
```

- For a segment with a non-unique key or no key:

If you want to insert an additional record for an existing key field, you must MATCH on the key field and specify ON MATCH INCLUDE.

- For segments with ACCESS=PE or MU, if a new occurrence is inserted, you must set the occurrence number (ORDER field) to 0 (zero indicates the next occurrence) or to a value greater than the number of existing occurrences. This new occurrence is always inserted after the last existing occurrence. For example, if a PE or MU segment has two existing occurrences, the next occurrence added will always be the third. If the occurrence with the given number already exists, processing will terminate with the following message:

```
(FOC4564) THIS OCCURRENCE ALREADY EXISTS. USE UPDATE COMMAND.
```

You should use the UPDATE command instead of INCLUDE in this case.

- For segments with ACCESS=MU, if the corresponding field with the MU option is defined *without* the NU option on the second level of the Adabas FDT (that is, it is part of a periodic group (PE) or simple group (GR)):
 - If an INCLUDE command is issued for the root segment alone, Adabas automatically inserts an empty child (MU) segment and, if this child segment has a parent with ACCESS=PE, an empty parent segment.

- If an INCLUDE command is issued for the root and a parent (PE) of the MU segment simultaneously, but without values for the child (MU) segment, Adabas automatically inserts an empty child (MU) segment. The data adapter automatically inserts the first occurrence of the parent (PE) segment using the values from the INCLUDE command.
- If an INCLUDE command is issued for the root, parent (PE), and child (MU) segments simultaneously, the data adapter inserts the first occurrence of both the child and parent segments using the values from the INCLUDE command.
- For segments with ACCESS=MU, if the corresponding field with the MU option is defined *with* the NU option on *any* level in the Adabas FDT, Adabas automatically suppresses the empty values. An INCLUDE command can be issued for the parent and child segments separately or simultaneously.

Note: The Master File does not identify the level of an MU field because if the MU field belongs to a simple group, the Master File does not describe this group. (However, if AUTOADBS created the Master File, there will be a commented declaration for the group.) Therefore, FOCUS cannot determine whether Adabas will add an empty occurrence for the corresponding segment. You must be aware of the options in the Adabas FDT to understand the behavior that occurs in response to an INCLUDE command for an MU segment.

Effect of UNQKEYNAME on INCLUDE Actions for Segments With ACCESS=ADBS

The UNQKEYNAME attribute in the Access File determines how the data adapter presents an INCLUDE command to Adabas. The option UQ in the Adabas FDT and the specific fields used in the MATCH command determine whether Adabas actually inserts the segment instance. The following table describes how these factors affect the result of the INCLUDE command. Assume that the Access File specifies UNQKEYNAME=EMPLOYEE_ID and that the employee id value EMPID005 already exists in the data source:

Result of INCLUDE Command for Existing EMPLOYEE_ID EMPID005

EMPLOYEE_ID has option UQ in FDT	Fields in MATCH command	Instance Inserted (ON NOMATCH)
No	EMPLOYEE_ID only	ON MATCH case performed
Yes	EMPLOYEE_ID only	ON MATCH case performed
EMPLOYEE_ID has option UQ in FDT	Fields in MATCH command	Instance Inserted (ON NOMATCH)
No	EMPLOYEE_ID plus fields with values that do not already exist	Yes
Yes	EMPLOYEE_ID plus fields with values that do not already exist	No - error (FOC4561), RC=198

This table describes INCLUDE actions when EMPLOYEE_ID is not in the MATCH field list:

Result of INCLUDE Command Without Matching on Field EMPLOYEE_ID

UNQKEYNAME = EMPLOYEE_ID?	Instance Inserted
Yes	No - error (FOC4563)
No	Yes

Reference Rules for Deleting Records From an Adabas Data Source

- For a segment with a unique key, the key field value is used to delete the target segment.
- For a segment with a non-unique key, you must supply the key field. If the MATCH criteria for this type of segment, in the path or target, do not identify a unique occurrence, the first occurrence found will be deleted. The use of Adabas descriptors for this type of segment is highly recommended for efficiency.
- For a segment with no key, you must supply at least one MATCH condition. If the MATCH criteria for this type of segment, in the path or target, do not identify a unique occurrence, the first occurrence found will be deleted.
- A segment occurrence with ACCESS=PE or MU is deleted from the data source except if it is the last occurrence for an ACCESS=PE segment. Adabas will only delete the last occurrence if all fields have the NU option in the FDT; if they do not all have this option, the occurrence will have empty values in all fields.

- When you delete segments that have dependent segments, the DELETED counter for the session may have an incorrect value. For a first level segment with descendants, this counter will always be incorrect. For a second level segment, this counter will be incorrect if there are multiple descendant segments. For a third level segment, this counter is always correct.

Reference Rules for Updating Records in an Adabas Data Source

- For a segment with a unique key, the key field value and any additional MATCH criteria are used to qualify the target segment for update.
- For a segment with a non-unique key, you must supply the key field. If the MATCH criteria for this type of segment, in the path or target, do not identify a unique occurrence, the first occurrence found will be updated. The use of Adabas descriptors for this type of segment is highly recommended for efficiency.
- For a segment with no key, you must supply at least one MATCH condition. If the MATCH criteria for this type of segment, in the path or target, do not identify a unique occurrence, the first occurrence found will be updated.

Example Updating Adabas Records With MODIFY

Suppose you want to display all the employees in a department and update the amount of leave they have taken:

```
MODIFY FILE EMPWRITE
CRTFORM LINE 1
" PLEASE ENTER A VALID DEPARTMENT </1"
1. " DEPARTMENT: <DEPARTMENT "
2. MATCH DEPARTMENT
   ON NOMATCH REJECT
   ON MATCH CRTFORM LINE 10
3. "ID: <D.EMPLOYEE_ID LEAVE DUE: <D.LEAVE_DUE> TAKEN <T.LEAVE_TAKEN> "
4. ON MATCH UPDATE LEAVE_TAKEN
   ON MATCH GOTO GETREST
   CASE GETREST
5. NEXT EMPLOYEE_ID
   ON NEXT CRTFORM LINE 10
   "ID: <D.EMPLOYEE_ID LEAVE DUE: <D.LEAVE_DUE> TAKEN <T.LEAVE_TAKEN> "
   ON NEXT UPDATE LEAVE_TAKEN
   ON NEXT GOTO GETREST
6. ON NONEXT GOTO EXIT
   ENDCASE
   DATA
   END
```

The MODIFY procedure processes as follows:

1. The user enters the department (DEPARTMENT) for the search, PROD.
2. The MATCH command causes Adabas to search the data source for the first record with the value PROD and return them in physical sequence. If none exists, the transaction is rejected.
3. If the supplied value matches a database value, the procedure displays it.

4. The procedure updates the LEAVE_TAKEN field for the first retrieved record using the turnaround value from the CRTFORM. EMPLOYEE_ID establishes the target record for the update.
5. Each time it executes the NEXT, the procedure retrieves the next record with the same department, PROD. It displays each one in physical order. It updates the LEAVE_TAKEN field for each retrieved record with the turnaround value.
6. When no more records exist for department PROD, the procedure ends.

Example Updating Adabas Records With Maintain

In Maintain, you can use stack columns as turnaround values to update a data source. The following annotated Maintain request updates the same records as the preceding MODIFY request:

```
MAINTAIN FILE EMPWRITE
INFER EMPLOYEE_ID LEAVE_DUE LEAVE_TAKEN INTO EMPSTACK
1. WINFORM SHOW WIN1
2. CASE MATCHREC
   FOR ALL NEXT EMPLOYEE_ID INTO EMPSTACK
   WHERE DEPARTMENT EQ VALSTACK.DEPARTMENT
   ENDCASE
3. CASE UPDLV
   FOR ALL UPDATE LEAVE_TAKEN FROM EMPSTACK
   ENDCASE
END
```

The Maintain procedure processes as follows:

1. A Winform named WIN1 displays. Assume that it displays an entry field labeled DEPARTMENT (whose source and destination stack is called VALSTACK) and a grid (scrollable data source) with columns EMPLOYEE_ID, LEAVE_DUE and LEAVE_TAKEN. See the *Maintaining Databases* manual for instructions on creating Winforms.

2. The user enters a DEPARTMENT value for the search and clicks the GetEmps button to invoke case MATCHREC. Case MATCHREC retrieves the records that satisfy the NEXT criteria and stores them in a stack named EMPSTACK. The Winform displays the retrieved records on the grid.
3. The user edits all the necessary leaves taken directly on the Winform grid and then clicks the Update button to invoke case UPDLV, which updates all leaves taken.

The following Winform displays when the GetEmp button (or PF4) is pressed with PROD entered in the Department field:

Department : PROD

EMPLOYEE_ID	LEAVE_DUE	LEAVE_TAKEN
11100305	31	12
11100308	31	5
11100309	30	28
11100310	30	5
11100311	30	10
11100312	29	28
11100313	30	0

GetEmp (PF4)

Update (PF6)

Exit (PF3)

The user can update the LEAVE_TAKEN field for all of the listed employees and update them all in one step with the Update button (or PF6).

Adabas Transaction Control Within MODIFY

The data adapter supports the Logical Unit of Work (LUW) concept. An LUW consists of one or more FOCUS maintenance actions (UPDATE, INCLUDE, or DELETE) that process as a single unit. The maintenance operations within the LUW can operate on the same or separate data sources.

In MODIFY, all records read by MATCH and NEXT commands are held by Adabas in a user's hold record queue. To prevent overflow of this queue, the user should periodically issue the SQL COMMIT WORK command to propagate the changes to the Adabas data source and clear the queue.

In Maintain, records are not held when they are read into a stack. They are held only in response to an updating command.

A transaction is defined as all actions taken since the application first accessed Adabas, last issued an SQL COMMIT WORK in MODIFY or COMMIT command in Maintain, or last issued an SQL ROLLBACK WORK in MODIFY or ROLLBACK command in Maintain.

Within a Logical Unit of Work, Adabas either executes all commands completely, or else it executes none of them. If Adabas detects no errors in any of the commands within the LUW:

- FOCUS issues a COMMIT WORK command. The data adapter issues an ET (End of Transaction) command to Adabas. The changes indicated by the updates within the transaction are recorded in the data source.
- Adabas releases locks on the target data.
- Database changes become available for other tasks.

In response to unsuccessful execution of any command in the transaction, the data adapter:

- Issues a BT (Backout Transaction) command to Adabas. Target data returns to its state prior to the unsuccessful transaction. All changes attempted by the commands in the transaction are backed out.
- Does not execute the remaining commands in the transaction.
- Releases locks on the target data.
- Discards partially accumulated results.

Adabas and the data adapter provide a level of automatic transaction management but, in many cases, this level of management alone is not sufficient. FOCUS supports explicit control of Adabas transactions with the commands SQL COMMIT WORK and SQL ROLLBACK WORK in MODIFY, and with the commands COMMIT and ROLLBACK in Maintain.

Note: SQL COMMIT WORK and SQL ROLLBACK WORK are data adapter commands. Do not confuse these commands with the FOCUS COMMIT WORK and ROLLBACK WORK commands that apply to FOCUS databases only. The data adapter ignores COMMIT WORK and ROLLBACK WORK without the SQL qualifier.

Unless you specify SQL COMMIT WORK and/or SQL ROLLBACK WORK in your MODIFY procedure (or COMMIT and/or ROLLBACK in your Maintain procedure), all FOCUS maintenance actions until the END command constitute a single LUW. If the procedure completes successfully, the data adapter automatically transmits an ET command to Adabas, and the changes become permanent. If the procedure terminates abnormally, the data adapter issues a BT command to Adabas, and the database remains untouched. Since locks are not released until the end of the program, a long MODIFY or Maintain procedure that relies on the default, end-of-program ET can interfere with concurrent access to data. In addition:

- You may lose all updates in the event of a system failure.

- You may fill up the hold queue Adabas establishes records locked by a user. The number of ISNs that can be held in this queue is determined by the Adabas NISNHQ parameter. To avoid exceeding this number, keep a counter in your MODIFY or Maintain procedure, and commit or rollback the transaction to avoid holding too many records.

Transaction Termination (COMMIT WORK)

The SQL COMMIT WORK command signals the successful completion of a transaction at the request of the procedure. Execution of a COMMIT command makes changes to the data sources permanent. The syntax in a MODIFY request is:

```
SQL COMMIT WORK
```

You can issue a COMMIT WORK as an ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT condition, after an update operation (INCLUDE, UPDATE, DELETE), or within cases of a case logic request.

Note: In Maintain, you must use the Maintain facility's COMMIT command to transmit an ET (End of Transaction) command to Adabas.

Example Using COMMIT WORK in a MODIFY Procedure

A COMMIT WORK example using Case Logic follows:

```
CASE PROCESS
  CRTFORM
  MATCH field1 ...
    ON MATCH insert, update, delete, ...
  GOTO EXACT
ENDCASE
CASE EXACT
  SQL COMMIT WORK
  GOTO TOP
ENDCASE
```

The PROCESS case handles the MATCH, ON MATCH, ON NOMATCH processing. Then it transfers to CASE EXACT, which commits the data instructing Adabas to write the entire Logical Unit of Work to the database.

Transaction Termination (ROLLBACK WORK)

The SQL ROLLBACK WORK command signals the unsuccessful completion of a transaction at the request of the procedure. Execution of a ROLLBACK command backs out all changes made to the data sources since the last COMMIT command. The syntax in a MODIFY request is:

SQL ROLLBACK WORK

You can design a MODIFY procedure to issue a ROLLBACK WORK command if you detect an error. For example, if a FOCUS VALIDATE test finds an inaccurate input value, you may choose to exit the transaction, backing out all changes since the last COMMIT. You can issue ROLLBACK WORK as an ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT condition, or within cases of a case logic request.

Note: In Maintain, you must use the Maintain facility's ROLLBACK command to transmit a BT command (Back out Transaction) to Adabas.

The data adapter automatically executes an SQL ROLLBACK WORK command when you exit from a transaction early. For example, if you exit a CRTFORM without specifying some action, the data adapter automatically issues a ROLLBACK WORK command on your behalf.

The data adapter automatically issues a BT command in case of system failure or when it detects a fatal data error, such as a reference to a field or data source that does not exist.

Example Using ROLLBACK WORK in a MODIFY Procedure

The following is an example of the ROLLBACK WORK command using Case Logic:

```
ON NOMATCH CRTFORM ...
ON NOMATCH VALIDATE ...
    ON INVALID GOTO ROLLCASE
    .
    .
    .
CASE ROLLCASE
    SQL ROLLBACK WORK
    GOTO TOP
ENDCASE
```

Code the ROLLBACK WORK command before a REJECT command. FOCUS ignores any action following the rejection of a transaction, except for GOTO or PERFORM.

For example:

```
ON MATCH SQL ROLLBACK WORK
ON MATCH REJECT
```

Example Transaction Control in Adabas

Each time an employee takes leave days, the following example updates the LEAVE_TAKEN field in the root segment of the EMPWRITE data source and posts a record for the leave start and end dates in the related AW0401 segment. To ensure that both updates complete or neither one does, the MODIFY procedure places both actions prior to a COMMIT WORK command. If the descendant data source is not processed, ROLLBACK WORK discards the whole logical transaction.

```
MODIFY FILE EMPWRITE
CRTFORM LINE 1
"/2 <25 MODIFY FOR LEAVE TAKEN </2 "
"<20 ENTER THE EMPLOYEE ID <EMPLOYEE_ID "
MATCH EMPLOYEE_ID
  ON MATCH CRTFORM LINE 7
  "<D.FIRST_NAME <D.LAST_NAME> LEAVE TAKEN <T.LEAVE_TAKEN> </1 "
  ON MATCH UPDATE LEAVE_TAKEN
  ON MATCH COMPUTE AW0401_OCC =0;
  ON NOMATCH REJECT
MATCH AW0401_OCC
  ON NOMATCH CRTFORM LINE 10
  "PLEASE ENTER LEAVE DATES"
  "LEAVE_START: <T.LEAVE_START> "
  "LEAVE_END: <T.LEAVE_END> "
  ON NOMATCH INCLUDE
  ON NOMATCH SQL COMMIT WORK
  ON MATCH SQL ROLLBACK WORK
  ON MATCH REJECT
DATA
END
```

Using the Return Code Variable: FOCERROR

FOCUS stores the return code from the updating commands INCLUDE, DELETE, and UPDATE in the variable FOCERROR:

- In MODIFY, the return code value is the Adabas response code.
- In Maintain, the value is
 - 1 if an INCLUDE command failed.
 - 2 if a DELETE command failed.
 - 3 if an UPDATE command failed.

A return code of zero indicates successful completion of the last updating command issued.

You can test the FOCERROR variable and take appropriate action if you encounter a non-fatal error. This condition might indicate the need to ROLLBACK the transaction or re-prompt the user for new input values. In Maintain, all errors after updating commands are non-fatal, and you should always test FOCERROR after issuing an updating command. In MODIFY, you can issue the SQL SET ERRORRUN ON command to make these errors non-fatal. If you do not issue this command, all errors after updating commands in MODIFY will be fatal.

All errors that result from retrieval commands such as MATCH or NEXT are fatal errors. An example of a command that causes a retrieval error is attempting to read a record held by another user. These errors terminate MODIFY and Maintain procedures.

For a list of common Adabas response codes, see [Adabas Response Codes](#).

Using the Data Adapter SET ERRORRUN Command

With SET ERRORRUN ON, MODIFY processing continues even when a serious error occurs, allowing applications to handle their own errors in the event that an Adabas error is part of the normal application flow. Code this command explicitly within the MODIFY procedure, preferably in CASE AT START, where it executes once.

Note: Maintain does not support the SET ERRORRUN command. All errors after updating commands are non-fatal.

When SET ERRORRUN is ON, the MODIFY procedure reports the error but continues execution. The MODIFY code can then test the value of FOCERROR to determine the cause of the error and take appropriate action. Be careful in evaluating the contents of FOCERROR, to prevent unpredictable errors in subsequent MODIFY processing.

SET ERRORRUN returns to its default setting of OFF at the end of the MODIFY procedure.

Syntax **How to Issue the SET ERRORRUN Command in a MODIFY Procedure**

```
CASE AT START  
    SQL SET ERRORRUN {OFF|ON}  
ENDCASE
```

where:

OFF

Stops MODIFY processing when the Data Adapter detects a fatal error. OFF is the default.

ON

Enables MODIFY processing to continue despite fatal errors. Test the value of FOCERROR to determine the desired action after an updating command fails. After the procedure ends, ERRORRUN returns to its default value of OFF.

Modifying Data sources Without Unique Keys

Adabas permits data sources with duplicate records. Such data sources cannot possibly have a unique key, since no combination of field values can make a given record unique.

The data adapter provides a way of maintaining data sources with duplicate records. However, only the first record encountered will be affected by data maintenance commands.

Note:

- Maintain does not support modifying unkeyed data sources.

Referential Integrity

The term *referential integrity* defines the type of consistency that should exist between parent and descendant segments.

- A parent segment must exist before a related record in a lower level segment can exist. For example, a specific employee ID must exist in the EMPWRITE data source before a salary can be added for that employee in the AQ0201 segment (INCLUDE referential integrity).
- If a parent segment is deleted, all of its descendant segments must be deleted (DELETE referential integrity).

FOCUS can provide referential integrity for the following types of data sources described in one multi-segment Master File:

- A single Adabas file.
- Multiple Adabas files connected by an embedded join in the Access File. The segment declarations for all such Adabas files must have ACCESS=ADBS in the Access File and must be joined using the Access File attributes KEYFLD and IXFLD. For more information on embedded joins, see the *FOCUS for IBM Mainframe ADABAS Interface User's Manual and Installation Guide*.

The following sections discuss referential integrity constraints.

FOCUS INCLUDE Referential Integrity

The FOCUS MODIFY facility syntax provides automatic referential integrity for inserting new records.

You must describe the data source in one multi-segment Master File. The multi-segment description establishes the relationship between the segments.

With a multi-segment Master File, you cannot add a descendant segment using the FOCUS MODIFY facility unless the parent segment already exists. Therefore, a MODIFY procedure that inserts records must MATCH on the parent segment before adding a record in descendant segment.

Example Using FOCUS INCLUDE Referential Integrity

The following examples demonstrate referential integrity when adding new records. The scenarios are:

1. Add a salary for an employee *only* if data for the employee ID already exists.
2. The employee ID does not exist. Add both a new employee ID and a salary.

A simple, annotated FOCUS MODIFY procedure for each scenario follows.

The first example adds course information only if a record already exists for the employee:

```
MODIFY FILE EMPWRITE
CRTFORM LINE 2
"ADD SALARY INFORMATION FOR EMPLOYEE  </1"
1. "EMPLOYEE ID: <EMPLOYEE_ID  </1 "
   "SALARY: <SALARY           CURR_CODE: <CURR_CODE "
2. MATCH EMPLOYEE_ID
   ON MATCH COMPUTE AQ0201_OCC = 0;
3. ON MATCH CONTINUE
4. ON NOMATCH REJECT
5. MATCH AQ0201_OCC
   ON NOMATCH INCLUDE
   ON MATCH REJECT
DATA
END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID, salary, and appropriate code. This constitutes the incoming transaction record.
2. The MATCH command causes Adabas to search the data source for an existing record with the specified employee ID.
3. If the employee record exists, the MODIFY sets the salary occurrence number to zero and continues to the next MATCH command.
4. If no record in the EMPWRITE data source exists with the specified employee ID, MODIFY rejects this transaction and routes control to the top of the procedure.
5. MATCH AQ0201_OCC causes Adabas to search for an existing salary record with the specified occurrence number for the employee ID located in Step 2. Because the occurrence number was set to zero in the previous MATCH command, no such record exists, and the MODIFY adds a new salary occurrence. If the occurrence number already existed, the MODIFY would reject the transaction as a duplicate.

The second example adds a record to the EMPWRITE data source for a new employee and adds a salary for that employee to the AQ0201 segment. If the employee ID already exists, the procedure adds only the salary information to the AQ0201 segment:

```
MODIFY FILE EMPWRITE
CRTFORM LINE 1
1. "ID: <EMPLOYEE_ID "
2. MATCH EMPLOYEE_ID
3. ON NOMATCH CRTFORM LINE 2
   "      LAST: <LAST_NAME      FIRST: <FIRST_NAME </1 "
   " MIDDLE: <MIDDLE_NAME> </1 "
   " ADDRESS_LINE_CNT <ADDRESS_LINE_CNT> CITY: <CITY "
   " ZIP: <ZIP_CODE      COUNTRY: <COUNTRY </1"
   " DEPT: <DEPT      INCOME_CNT: <INCOME_CNT </1"
   "LEAVE DUE: <LEAVE_DUE      DEPARTMENT: <DEPARTMENT "
   "SALARY: <SALARY      CODE: <CURR_CODE "
ON NOMATCH INCLUDE
ON NOMATCH COMPUTE AQ0201_OCC = 0;
ON MATCH COMPUTE AQ0201_OCC = 0;
4. ON MATCH CRTFORM LINE 9
   "SALARY: <SALARY      CODE: <CURR_CODE "R_TAKEN QTR: <QTR "
5. MATCH AQ0201_OCC
ON NOMATCH INCLUDE
ON MATCH REJECT
DATA
END
```

The MODIFY procedure processes as follows:

1. The user enters EMPLOYEE_ID.
2. The MATCH command causes Adabas to search the EMPWRITE data source for an existing record for the specified employee ID.
3. If the employee record does not exist, the user enters the data for both the employee and the salary. The procedure adds a record to each segment.

4. If the employee already exists, the user enters only the salary data.
5. The MATCH AQ0201_OCC command causes Adabas to search the AQ0201 segment for the specified occurrence. If this occurrence does not exist for this employee, the procedure adds it. If it does exist, the procedure rejects the transaction.

FOCUS DELETE Referential Integrity

FOCUS provides automatic referential integrity for deleting records described in a multi-segment Master File. Just as with INCLUDE referential integrity, only data sources described in a multi-segment Master File invoke FOCUS DELETE referential integrity.

When you delete a parent segment in a MODIFY or Maintain procedure, FOCUS automatically deletes all descendant segments at the same time.

Example Using FOCUS DELETE Referential Integrity

For example, when you delete an employee from the root segment in the EMPWRITE Master File, FOCUS also deletes all records from the descendent segments for the employee:

```
MODIFY FILE EMPWRITE
CRTFORM LINE 2
"DELETE EMPLOYEE </1"
1. "EMPLOYEE ID: <EMPLOYEE_ID   "
2. MATCH EMPLOYEE_ID
   ON MATCH COMPUTE DOIT/A1 = 'N';
   ON MATCH CRTFORM LINE 6
3. "EMPLOYEE TO BE DELETED: <D.EMPLOYEE_ID </1"
   "          LAST NAME:   <D.LAST_NAME </1"
   "          FIRST NAME:  <D.FIRST_NAME </1"
   "          DEPARTMENT:  <D.DEPARTMENT </1"
   "IS THIS THE EMPLOYEE YOU WISH TO DELETE? (Y,N): <DOIT "
   ON MATCH IF DOIT EQ 'N' THEN GOTO TOP;
4. ON MATCH DELETE
   ON NOMATCH REJECT
   DATA
   END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID.
2. The MATCH command causes Adabas to search the root segment of the EMPWRITE data source for an existing record with the specified employee ID.
3. If the record exists, the MODIFY displays information for verification.
4. Once verified, FOCUS deletes the employee and all associated segment instances. When FOCUS deletes a parent instance, it automatically deletes all associated related instances.

Inhibiting FOCUS Referential Integrity

You may not always want to enforce FOCUS referential integrity. If the referential integrity is being enforced between separate Adabas files described in a single Master File using an embedded join described in the Access File, you can describe each file you want to modify separately in a separate Master File.

Another technique is to COMBINE data sources rather than using a multi-segment Master File. COMBINE of single data sources does not invoke FOCUS referential integrity.

The MODIFY COMBINE Facility

Some applications require that you use a single input transaction to update several data sources in the same MODIFY procedure. If the data sources are not defined in the same Master File, you can use the COMBINE facility to modify them as if they are one.

Note: In Maintain, you do not issue a COMBINE command to modify unrelated data sources. Instead, you reference multiple data sources in the MAINTAIN FILE command. For example:

```
MAINTAIN FILES EMPWRITE AND COURSE
```

You can maintain up to 63 data sources in a single MODIFY procedure that operates on a COMBINE structure. The COMBINE limit is 16 Master Files; however, each Master File can describe more than one data source, for a total of 64 per procedure, minus one for the virtual root segment created by the COMBINE command.

The COMBINE facility links multiple data sources and assigns a new name to them so FOCUS can treat the data sources as a single structure. Data sources in a COMBINE structure can have different SUFFIX attributes, but you cannot combine a FOCUS database with anything except other FOCUS databases.

Note: In Maintain, you can modify FOCUS databases and Adabas data sources in the same procedure.

When you issue a COMBINE command, the COMBINE structure remains in effect for the duration of the FOCUS session or until you enter another COMBINE command. Only one COMBINE structure can exist at a time, so each subsequent COMBINE command replaces the existing structure.

Do not confuse COMBINE with the dynamic JOIN command. You use JOIN to *report* from multiple data sources that share at least one common field or for LOOKUP functions. With the COMBINE facility, you can *MODIFY* multiple data sources. COMBINE is part of the MODIFY command; only the MODIFY and CHECK FILE commands process COMBINE structures. The FIND function also works in conjunction with COMBINE (see [The FIND Function](#)).

Note that COMBINE considers the component structures to be unrelated. FOCUS referential integrity does not apply to a COMBINE of single-data source Master Files. Your procedure should check for and enforce referential integrity, if necessary.

Syntax How to Use the COMBINE Command

The basic syntax for the COMBINE command is

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

where:

file1 - *filen*

Are the Master File names of the data sources you want to modify. You can specify up to 16 Master Files.

pref1 - prefn

Are prefix strings for each file; up to four characters. They provide uniqueness for fieldnames. You cannot mix TAG and PREFIX in a COMBINE structure. Refer to the FOCUS for S/390 documentation for additional information.

tag1 - tagn

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

AND

Is an optional word to enhance readability.

asname

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

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

How FOCUS Creates a COMBINE Structure

For example, the EMPINFO data source contains employee number, last name, first name, hire date, department code, current job code, current salary, number of education hours, and bonus plan information. A second data source, PAYINFO, is a historical record of the employee's pay history. It contains the employee number, date of increase, percent of increase, new salary, and job code.

Each time a salary changes, both the EMPINFO and PAYINFO data sources must reflect the change. Since both data sources need to share data entered for employee number, salary and job code, this application is appropriate for the COMBINE facility. You can update both data sources at the same time without having to define multi-segment Master and Access Files.

The following figures represent the data sources as separate entities.

EMPINFO data source

```
EMPINFO
01      S0
*****
*EMPLOYEE_ID **
*LAST_NAME   **
*FIRST_NAME  **
*HIRE_DATE   **
*           **
*****
*****
```

EMPINFO

PAYINFO data source

```
PAYINFO
01      S0
*****
*PAYEID      **
*DAT_INC     **
*PCT_INC     **
*SALARY      **
*           **
*****
*****
```

PAYINFO

To modify the data sources simultaneously, issue the following sequence of commands at the FOCUS command level or in a FOCEXEC:

```
COMBINE FILES EMPINFO PAYINFO AS EMPSPAY
MODIFY FILE EMPSPAY
```

.
.
.

In the following picture, generated by the CHECK FILE command, FOCUS defines a new segment, identified as SYSTEM99, to be the root segment of the combined structure.

SYSTEM99 acts as the traffic controller for this structure; it is a virtual (artificial) segment. It counts as one segment towards the total of 64 segments allowed in the COMBINE structure.

check file empspay pict

```
NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=   3 ( REAL=   3 VIRTUAL=   0 )
NUMBER OF FIELDS=   15 INDEXES=   0 FILES=   3
TOTAL LENGTH OF ALL FIELDS=  95
```

SECTION 01

STRUCTURE OF SQLDS FILE EMPINFO ON 07/22/93 AT 09.54.27

```
          SYSTEM99
01      S0
*****
*          **
*          **
*          **
*          **
*          **
*****
*****
          I
          +-----+
          I          I
          I EMPINFO      I PAYINFO
02      I S0          03      I S0
*****          *****
*EMPLOYEE_ID ** *PAYEID      **
*LAST_NAME   ** *DAT_INC    **
*FIRST_NAME  ** *PCT_INC    **
*HIRE_DATE   ** *SALARY     **
*           ** *           **
*****          *****
*****          *****
          EMPINFO          PAYINFO
```

The COMBINE facility makes it easy to modify many files with the same transaction. For additional information regarding the COMBINE facility of MODIFY, refer to the *Maintaining Databases* manual.

The LOOKUP Function

The LOOKUP function, used in FOCUS MODIFY procedures, retrieves data values from cross-referenced data sources joined dynamically with the JOIN command. The function is valid in both MODIFY COMPUTE and VALIDATE commands.

The syntax for the LOOKUP function is

```
rfield/I1 = LOOKUP(field);
```

where:

rfield

Contains the return code (1 or 0) after the LOOKUP function executes.

field

Is the name of any field in a cross-referenced data source. After the LOOKUP, this fieldname contains the field's value for you to use as needed.

To use this feature most efficiently with Adabas, specify a cross-referenced field for which an Adabas descriptor has been established.

Note:

- The LOOKUP function is not supported between Adabas data sources and FOCUS databases in either direction.
- The extended syntax of the LOOKUP function (parameters GE and LE) is not valid for Adabas data sources. LOOKUP can only retrieve values that match exactly. Refer to the FOCUS for S/390 documentation for more information.

The FIND Function

The FIND function, used with COMBINE structures in FOCUS MODIFY procedures or with any file in a Maintain procedure, verifies the existence of a value in another data source. The FIND function sets a temporary field to 1 if the value exists in the other data source and to 0 if it does not. FIND does not return any actual data values.

Use FIND only with a data source referenced in a COMBINE command or a MAINTAIN FILE command. With COMBINE, if the FIND is for a field in a VSAM file, this field must be the index or alternate index field. For Maintain, the field must be indexed only if it is in a FOCUS database.

The syntax for the FIND function is

```
rfield/I1 = FIND(fieldname AS dbfield IN file);
```

where:

rfield

Contains the return code (1 or 0) after the FIND function executes.

fieldname

Is the comparison field from one COMBINE data source or one data source referenced in a MAINTAIN FILE command.

dbfield

For MODIFY, is the field in another COMBINE file structure or an indexed field in a VSAM file, to use for the value comparison. The AS *dbfield* clause is optional if *rfield* and *dbfield* have the same name.

For Maintain, is a *fieldname* from one of the files listed in the MAINTAIN FILE command, qualified with its file name.

To use this feature most efficiently with Adabas, specify a field for which an Adabas descriptor has been established.

file

In MODIFY, names the data source or VSAM file in which dbfield resides. In Maintain, is ignored.

The FIND function is only supported within MODIFY or Maintain procedures. For more information, consult the FOCUS for S/390 documentation.

Data Adapter Error Messages and Adabas Response Codes

This section lists the Adabas Write Data Adapter error messages and common Adabas response codes.

Adabas Write Data Adapter Error Messages

In the following error messages, the term *number* refers to the response code that Adabas returns. For example, the following message references Adabas error response code 113 in the Adabas Messages and Codes manual:

(FOC4561) ERROR IN ADABAS INCLUDE/113

The following messages apply to the Adabas Write Data Adapter:

(FOC4555) INVALID UNQKEYNAME FOR SEGMENT
(FOC4557) ERROR IN ADABAS COMMIT/*number*
(FOC4558) ERROR IN ADABAS ROLLBACK/*number*
(FOC4559) ERROR IN ADABAS UPDATE/*number*
(FOC4560) ERROR IN ADABAS DELETE/*number*
(FOC4561) ERROR IN ADABAS INCLUDE/*number*
(FOC4562) RECORD IS HELD BY ANOTHER USER
(FOC4563) INCORRECT INCLUDE REQUEST FOR NON UNIQUE KEY
(FOC4564) THIS OCCURRENCE ALREADY EXISTS. USE UPDATE COMMAND.

(FOC4565) IGNORED ATTEMPT TO CHANGE NONUPDATABLE FIELD

An UPDATE command was used against a nonupdatable field,
i.e. - field described in AFD as SUB/SUPERDESCRIPTOR or part
of it;
- field that refers in MFD to the same DB field (ALIAS) as another
field (synonym case).

For individual Adabas Data Adapter error messages, consult Appendix B,
Adabas Interface Error Messages in the FOCUS for IBM Mainframe Adabas
Interface User's Manual and Installation Guide.

Adabas Response Codes

FOCUS Error	Adabas Response Code	Description
FOC4490	148	The database is down or unavailable.
FOC4496	148	Check the DBNO in the Access File.
FOC4500	113	The FILENO for the segment in the Access File is not correct.
FOC4504	0	Check the DBNO and FILENO in both the PREDDDB and PREDEL Access Files.
FOC4504	17	The database contains no data and/or the database does not exist in the Predict dictionary.

FOCUS Error	Adabas Response Code	Description
FOC4504	28	Check the compatibility of the Predict dictionary and FDT listing.
FOC4504	41	The Predict dictionary and FDT may not match. Check the group in the FDT. It may have been created differently in the Predict dictionary.
FOC4504	201	The password is not correct.
Any	9	As a rule: The transaction time limit (TT) or the transaction non-activity time limit (TNAE, TNAX, TNAA) has been exceeded. For details see a subcode value in the Adabas control block's ADD2 field in the data adapter's level 3 trace, and refer to its explanation in the Adabas <i>Messages and Codes</i> manual.
Any	47	The maximum value for the NISNHQ parameter was exceeded. Too many records are held. You can release them by periodically issuing the COMMIT command.

FOCUS Error	Adabas Response Code	Description
Any	40-45	One or more errors occurred in the Format Buffer. There is a discrepancy between the Master File and the Adabas FDT.
Any	50-59	One or more errors occurred in the Record Buffer. There is a discrepancy between the Master File and the Adabas FDT.
Any	145	An attempt was made to hold a record already in the hold queue for another user.

If variables FOCERROR or RETCODE have the value 999, the Adabas response code was not used. Use the data adapter error message to determine the cause of the error.

For details on the Adabas response codes that may accompany the Adabas Data Adapter error messages, see your Software AG publication, *General Reference Set: Messages and Codes Manual*. If a subcode details the specific cause and actions for the response code in this manual, you can find the subcode's value in the low-order (rightmost) two bytes of the Adabas control block's ADD2 field using the Adabas Data Adapter's level 3 trace. The trace facility for the Write data adapter is the same as for the Read data adapter.

Common User Errors

1. Using a group field in an INCLUDE or UPDATE command.

In the following request, the record is not inserted because the field list contains a group field (LEAVE_DATA):

```
MODIFY FILE EMPWRITE
-* This is an example of the error caused by using a
-* group field (LEAVE_DATA), rather than only elementary fields,
-* in an INCLUDE process.
-* The insert (Segname=S01) is rejected with the error -
-* (FOC4491) FIND ERROR READING FIRST RECORD : S01 /61
-*
-* Insert for SEGNAM=S01 - ACCESS=ADBS
COMPUTE EMPLOYEE_ID = 'EMPID003';
MATCH EMPLOYEE_ID
ON MATCH COMPUTE
    LAST_NAME = 'SMITH' ;
    FIRST_NAME = 'ROBERT' ;
    MIDDLE_NAME = 'EDWARD' ;
    CITY = 'NEW YORK' ;
    ZIP_CODE = '10121' ;
    COUNTRY = 'USA' ;
    LEAVE_DATA = 'VAC0801080700';
ON MATCH INCLUDE
ON NOMATCH REJECT
DATA
END
```

Solution: Use only elementary fields in the field list for INCLUDE and UPDATE.

2. Issuing an INCLUDE command for a new key value with additional fields in the MATCH list.

In the following example EMPLOYEE_ID is defined as a unique key in the Access File. However, Adabas may insert the record even if the employee ID value already exists in the data source:

```
MODIFY FILE EMPWRITE

- * INSERT FOR SEGNAM=S01 - ACCESS=ADBS
  COMPUTE EMPLOYEE_ID = 'EMPID001';
    LAST_NAME = 'SMITH';
MATCH EMPLOYEE_ID LAST_NAME
ON NOMATCH COMPUTE
  FIRST_NAME = 'ROBERT';
  MNAME      = 'EDWARD';
  CITY       = 'NEW YORK';
ON NOMATCH INCLUDE
ON MATCH REJECT

DATA
END
```

Solution: Issue the MATCH command for the field EMPLOYEE_ID alone, then enter values into the other fields and issue the INCLUDE command.

7.0.9 New Features

Fusion

[NF575: Fusion](#)

General Enhancements

[NF716: Euro Currency Support](#)

[NF655: FOCPROF - The System Wide Profile](#)

[NF735: Enhancement to ? SET](#)

[NF746: Leading Zeros](#)

[NF656: Controlling REBUILD Messages](#)

[NF740: Changes to the REBUILD Prompt](#)

[NF660: Multi-volume Support in MVS FOCUS](#)

[NF670: DYNAM Support for Unit Count](#)

[NF718: DYNAM Support for Existing Relative GDG
Numbers](#)

[NF745: ? PTF Enhancements](#)

Reporting Enhancements

[NF691: Escape Character for LIKE Predicate](#)

[NF744: HOLD FORMAT EXCEL](#)

[NF748: HOLD FORMAT WP With Carriage Control](#)

Performance Enhancements

[NF654: HOLD From External Sort](#)

[NF597: Aggregation by External Sort](#)

[NF728: Changing Retrieval Order with Aggregation](#)

Web Interface for FOCUS

[NF683: Web Interface support for Maintain Winforms](#)

[NF684: PCHOLD for Non-Html Files](#)

[NF730: Hold Format PDF](#)

Relational Interfaces

[NF720: SQLJOIN OUTER Setting for Relational Interfaces](#)

Teradata Interface

[NF652: Teradata Interface Kanji Support](#)

Model 204 Interface

[NF673: Model 204 Interface Account Split](#)

CA-IDMS Interface

[NF584: Dynamically Setting the IDMS DBNAME and
DICTNAME](#)

FOCUS Client

[NF722: FOCUS Client DNS Names Support](#)

NF575: Fusion

FOCUS Version 7.0 release 9 introduces the Fusion database for CMS and MVS FOCUS. Fusion is a high performance database whose unique Multi Dimensional Indexing (MDI) architecture extends the scope of high-speed multi-dimensional query performance.

For complete details, see the *Fusion User's Manual for EDA 4.2 and FOCUS 7.0* (DN3700041.1198).

NF716: Euro Currency Support

With the introduction of the euro currency, businesses need to maintain books in two currencies, add new fields to their database designs, and perform new types of currency conversions. This new feature gives FOCUS the ability to perform currency conversions according to the rules specified by the European Union. Before you can use FOCUS to process currency conversions, you must:

- Create a currency database with the currency IDs and exchange rates you will use. See [Creating the Currency Database](#).
- Identify fields in your data sources that represent currency data. See [Identifying Fields That Contain Currency Data](#).
- Activate your currency database. See [Activating the Currency Database](#).

After you complete these preliminary steps, you can perform currency conversions. See [Processing Currency Data](#).

Note: Operating system vendors are in the process of integrating the euro currency symbol into their environments. As the euro symbol becomes available, FOCUS will support it.

Converting Currencies

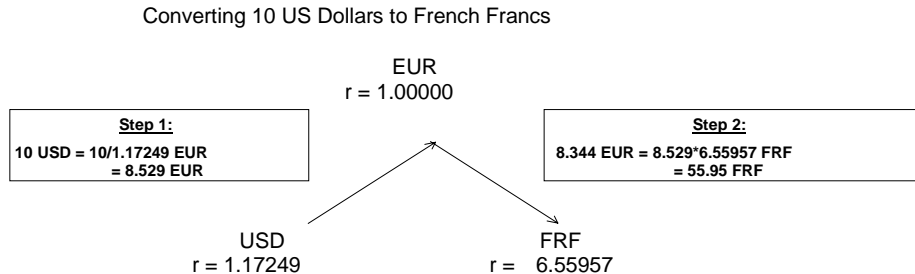
Although the euro was introduced in 11 countries of the European Union on January 1, 1999, it will not immediately replace local currencies in those countries. During the transition period from 1999 to 2002, both traditional currencies and the euro will be used simultaneously for accounting purposes and non-cash transactions in each participating country. Euro cash will not be introduced until January 1, 2002, and by July 1, 2002 the traditional currencies will no longer be legal tender.

On the last day of 1998, the European Union set fixed exchange rates between the euro and the traditional national currency in each of the 11 adopting member nations. While the exchange rates within “Euroland” will remain fixed, exchange rates between the euro and non-euro countries will continue to vary freely and, in fact, several rates may be in use at one time (for example, actual and budgeted rates).

The European Union has established the following rules for currency conversion:

- The exchange rate must be specified as a decimal value, r , with six significant digits (not six decimal places). For example, 123.456 has six significant digits but not six decimal places. This rate will establish the following relationship between the euro and the particular national currency:
1 euro = r national units
- To convert from the euro to the national unit, multiply by r and round the result to two decimal places.
- To convert from the national currency to the euro, divide by r and round the result to two decimal places.

- To convert from one national currency to another, first convert from one national unit to the euro, rounding the result to at least three decimal places (FOCUS rounds to exactly three decimal places). Then convert from the euro to the second national unit, rounding the result to two decimal places. The following diagram illustrates this two-step conversion process known as *triangulation*:



Preparing FOCUS to Process Currency Conversions

Although 11 or more currencies in the European Union will be converting to the euro, more than 100 currencies have a recognized status worldwide. In addition, you may need to define custom currencies for some applications. You identify your currency codes and rates by creating a currency database. The currency database can be any type of data source that FOCUS can access.

Creating the Currency Database

For each type of currency you need, you must supply the following values in your currency database:

- A three-character code to identify the currency, such as USD for U.S. dollars or BEF for Belgian francs. (For a partial list of recognized currency codes, see [Sample Currency Codes](#).)
- One or more exchange rates for the currency.

There is no limit to the number of currencies you can add to your currency database; the currencies you can define are not limited to official currencies and, therefore, the currency database can be fully customized for your applications.

We strongly recommend that you create a separate database for the currency data rather than adding the currency fields to another data source. A separate currency database enhances performance and minimizes resource utilization because FOCUS loads the currency database into memory before you perform currency conversions.

Syntax How to Specify Currency Codes and Rates in a Master File

The currency database can be any type of data source accessible by FOCUS (for example, FOCUS, FIX, DB2, or VSAM). The currency Master File must have one field that identifies each currency ID you will use and one or more fields to specify the exchange rates.

The syntax is

```
FIELD = CURRENCY_ID,      FORMAT = A3,                ACTUAL = A3 , $
FIELD = rate1,            FORMAT = {D12.6|numeric_format1}, ACTUAL = A12,$
.
.
.
FIELD = raten,            FORMAT = {D12.6|numeric_formatn}, ACTUAL = A12,$
```

where:

`CURRENCY_ID`

Is the required field name. The values stored in this field are the three-character codes that identify each currency, such as USD for U.S. dollars. Each currency ID can be a universally recognized code or a user-defined code. **Note:** FOCUS automatically recognizes the code EUR; you should *not* store this code in your currency database. See [Sample Currency Codes](#) for a list of common currency codes.

`rate1,...,raten`

Are types of rates (such as BUDGET, FASB, ACTUAL) to be used in currency conversions. Each rate is the number of national units that represent one euro.

`numeric_format1,...,numeric_formatn`

Are the display formats for the exchange rates. Each format must be numeric. The recommended format, D12.6, ensures that the rate is expressed with six significant digits as required by the European Union conversion rules. Do not use Integer format (I).

`ACTUAL An`

Is required only for non-FOCUS data sources.

Note: The maximum number of fields in the currency database must not exceed 255 (that is, the CURRENCY_ID field plus 254 currency conversion fields).

Example Specifying Currency Codes and Rates in a Master File

The following Master File for a comma-delimited currency database specifies two rates for each currency, ACTUAL and BUDGET:

```
FILE = CURRCODE, SUFFIX = COM,$
FIELD = CURRENCY_ID,      FORMAT = A3,    ACTUAL = A3 ,,$
FIELD = ACTUAL, ALIAS = ,  FORMAT = D12.6, ACTUAL = A12 ,,$
FIELD = BUDGET, ALIAS = ,  FORMAT = D12.6, ACTUAL = A12 ,,$
```

The following is sample data for the currency database defined by this Master File:

```
FRF, 6.55957, 6.50000,$
USD, 1.17249, 1.20000,$
BEF, 40.3399, 41.00000,$
```

Identifying Fields That Contain Currency Data

Once you have created your currency database, you must identify the fields in your data sources that represent currency values. To designate a field as a currency-denominated value (a value that represents a number of units in a specific type of currency) add the CURRENCY attribute to one of the following:

- The FIELD specification in the Master File.
- The left side of a DEFINE or COMPUTE.

Syntax How to Identify a Currency Value

Use the following syntax to identify a currency-denominated value:

- In a Master File

```
FIELD = currfield, FORMAT = numeric_format, ,
CURR = {curr_id|codefield} ,,$
```

- In a DEFINE in the Master File

```
DEFINE currfield/numeric_format CURR curr_id = expression ;$
```

- In a DEFINE FILE command

```
DEFINE FILE filename  
currfield/numeric_format CURR curr_id = expression ;  
END
```

- In a COMPUTE command

```
COMPUTE currfield/numeric_format CURR curr_id = expression ;
```

where:

filename

Is the name of the file for which this field is defined.

currfield

Is the name of the currency-denominated field.

numeric_format

Is a numeric format. Depending on the currency denomination involved, the recommended number of decimal places is either two or zero. Do not use I or F format.

CURR

Indicates that the field value represents a currency-denominated value. CURR is an abbreviation of CURRENCY, which is the full attribute name.

curr_id

Is the three-character currency ID associated with the field. In order to perform currency conversions, this ID must either be the value EUR or match a CURRENCY_ID value in your currency database.

codefield

Is the name of a field, qualified if necessary, that contains the currency ID associated with *currfield*. The code field should have format A3 or longer and is interpreted as containing the currency ID value in its first three bytes.

For example:

```
FIELD = PRICE, FORMAT = P12.2C, ..., CURR = TABLE.FLD1,$  
.  
.  
.  
FIELD = FLD1, FORMAT = A3, ..., $
```

The field named FLD1 contains the currency ID for the field named PRICE.

expression

Is a valid expression.

Example Identifying a Currency-denominated Field

Assume that the currency database contains the currency ID value BEF (Belgian francs).

If the FINANCE data source contains a field named PRICE that is denominated in Belgian francs, the description of PRICE in the FINANCE Master File could be:

```
FIELD = PRICE, ALIAS=, FORMAT = P17.2, CURR=BEF,$
```

Activating the Currency Database

Before you can perform currency conversions, you must bring the relevant currency database into memory by issuing the SET EUROFILE command.

Syntax How to Activate Your Currency Database

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

```
SET EUROFILE = {ddname | OFF}
```

where:

ddname

Is the name of the Master File for the currency database. There is no default value for EUROFILE. The ddname must refer to a data source known to FOCUS and accessible by FOCUS in read-only mode.

OFF

Deactivates the currency database and removes it from memory.

During your FOCUS session, if you want to access a different currency database, you can re-issue the SET EUROFILE command.

Note:

- You cannot append any additional SET parameters to the SET EUROFILE command line. For example, the PAUSE setting would be lost if you issued the following command:

```
SET EUROFILE=filename , PAUSE=OFF
```

- You cannot issue the SET EUROFILE command within a TABLE request.

Syntax How to Determine the Currency Database in Effect

If you want to determine which currency database is in effect, issue the ? SET ALL command or the new EUROFILE query command:

```
? SET EUROFILE
```

Example Determining the Currency Database in Effect

Assume the currency database is named CURRCODE.

If you issue the following commands:

```
set eurofile = currcode  
? set eurofile
```


FOCUS returns the following response:

EUROFILE

CURRCODE

Reference SET EUROFILE Error Messages and Notes

Issuing the SET EUROFILE command when the currency database Master File does not exist generates the following error message:

```
(FOC205) THE DESCRIPTION CANNOT BE FOUND FOR FILE NAMED: ddname
```

Issuing the SET EUROFILE command when the currency Master File specifies a FOCUS database and the associated FOCUS database does not exist generates the following error message:

```
(FOC036) NO DATA FOUND FOR THE FOCUS FILE NAMED: name
```

Note for Pooled Table users: The SET EUROFILE command creates a pool boundary.

Processing Currency Data

After you have created your currency database, identified the currency-denominated fields in your data sources, and activated your currency database, you can perform currency conversions.

Each currency ID in your currency database generates a virtual conversion function whose name is the same as its currency ID. For example, if you added BEF to your currency database, a virtual BEF currency conversion function will be generated.

The euro function, EUR, is supplied automatically with FOCUS. You do not need to add the EUR currency ID to your currency database.

Syntax How to Convert Currency Data

Use the following syntax for calling a currency conversion function

- In a TABLE, GRAPH, or MODIFY procedure:

```
DEFINE FILE filename  
result/format [CURR curr_id] = curr_id(infield, rate1 [,rate2]);  
END
```

or

```
COMPUTE result/format [CURR curr_id] = curr_id(infield, rate1  
[,rate2]);
```

- In a Master File:

```
DEFINE result/format [CURR curr_id] = curr_id(infield, rate1  
[,rate2]);$
```

where:

filename

Is the name of the file for which this field is defined.

result

Is the converted currency value.

format

Must be a numeric format. Depending on the currency denomination involved, the recommended number of decimal places is either two or zero. Do not use I or F format. The result will always be rounded to two decimal places, which will display if the format allows at least two decimal places.

curr_id

Is the currency ID of the result field. This ID must be the value EUR or match a currency ID in your currency database; any other value generates the following message

```
(FOC263) EXTERNAL FUNCTION OR LOAD MODULE NOT FOUND: curr_id
```

Note: The CURR attribute on the left side of the DEFINE or COMPUTE identifies the result field as a currency-denominated value which can be passed as an argument to a currency function in subsequent currency calculations. Adding this attribute to the left side of the DEFINE or COMPUTE does not invoke any format or value conversion on the calculated result.

infield

Is a currency-denominated value. This input value will be converted from its original currency to the *curr_id* denomination. If the *infield* and *result* currencies are the same, no calculation is performed and the *result* value is the same as the *infield* value.

rate1

Is the name of a rate field from the currency database. The *infield* value is divided by its currency's *rate1* value to produce the equivalent number of euros.

If *rate2* is not specified in the currency calculation and triangulation is required, this intermediate result is then multiplied by the *result* currency's *rate1* value to complete the conversion.

In certain cases, you may need to provide different rates for special purposes. In these situations you can specify any field or numeric constant for *rate1* as long as it indicates the number of units of the *infield* currency denomination that equals one euro.

rate2

Is the name of a rate field from the currency database. This argument is only used for those cases of triangulation in which you need to specify different rate fields for the *infield* and *result* currencies. It is ignored if the euro is one of the currencies involved in the calculation.

The number of euros that was derived using *rate1* is multiplied by the *result* currency's *rate2* value to complete the conversion.

In certain cases, you may need to provide different rates for special purposes. In these situations you can specify any field or numeric constant for *rate2* as long as it indicates the number of units of the *result* currency denomination that equals one euro.

Note: MAINTAIN does not support these currency conversion functions.

Example Converting Currencies

Assume that the currency database contains the currency IDs USD and BEF, and that PRICE is denominated in Belgian francs as follows:

```
FIELD = PRICE, ALIAS=, FORMAT = P17.2, CURR=BEF,$
```

- The following example converts PRICE to euros and stores the result in PRICE2 using the BUDGET conversion rate for the BEF currency ID:

```
COMPUTE PRICE2/P17.2 CURR EUR = EUR(PRICE, BUDGET);
```

- This example converts PRICE from Belgian francs to US dollars using the triangulation rule:

```
DEFINE PRICE3/P17.2 CURR USD = USD(PRICE, ACTUAL);$
```

First PRICE is divided by the ACTUAL rate for Belgian francs to derive the number of euros rounded to three decimal places. Then this intermediate value is multiplied by the ACTUAL rate for US dollars and rounded to two decimal places.

- The following example uses a numeric constant for the conversion rate:

```
DEFINE PRICE4/P17.2 CURR EUR = EUR(PRICE,5);$
```

- The next example uses the ACTUAL rate for Belgian francs in the division and the BUDGET rate for US dollars in the multiplication:

```
DEFINE PRICE5/P17.2 CURR USD = USD(PRICE, ACTUAL, BUDGET);$
```

Reference Currency Calculation Processing and Messages

The result is always calculated with very high precision, 31 to 36 significant digits, depending on platform. The precision of the final result is always rounded to two decimal places. In order to display the result to the proper precision, its format must allow at least two decimal places.

Issuing a TABLE request against a Master File that specifies a currency code not listed in the active currency database generates the following message:

`(FOC1911) CURRENCY IN FILE DESCRIPTION NOT FOUND IN DATA`

A syntax error or undefined fieldname in a currency conversion expression generates the following message:

`(FOC1912) ERROR IN PARSING CURRENCY STATEMENT`

Reference Sample Currency Codes

The following rates were in effect on December 31, 1998. Euroland countries as of that date are marked with an asterisk (*). Their rates are fixed and will not change; the rates for other countries can change over time:

Country	Currency Code	Rate
Austria*	ATS	13.7603
Belgium*	BEF	40.3399
Canada	CAD	1.7978
Denmark	DKK	7.46215
European Union	EUR	1
Finland*	FIM	5.94573

Country	Currency Code	Rate
France*	FRF	6.55957
Germany*	DEM	1.95583
Greece	GRD	328.6
Ireland*	IEP	0.787564
Italy*	ITL	1936.27
Japan	JPY	133.149
Luxembourg*	LUF	40.3399
Netherlands*	NLG	2.20371
Norway	NOK	8.91039
Portugal*	PTE	200.482
Spain*	ESP	166.386
Sweden	SEK	9.52669
Switzerland	CHF	1.61093
UK	GBP	0.706739
USA	USD	1.17249

Example Converting U.S. Dollars to Euros, French Francs, and Belgian Francs

Assume PRICE is denominated in U.S. dollars and ACTUAL is the name of a rate in the currency database. Using the currency conversion rates from [Sample Currency Codes](#), the following FOCEXEC converts PRICE to euros, French francs, and Belgian francs:

```
-* CURRCODE IS THE CURRENCY DATABASE
-* CURRDATA IS THE DATA SOURCE WITH CURRENCY-DENOMINATED FIELDS

-* THE FOLLOWING FILEDEFS ARE FOR RUNNING UNDER CMS
CMS FILEDEF CURRCODE DISK CURRCODE TEXT A
CMS FILEDEF CURRDATA DISK CURRDATA TEXT A

-* THE FOLLOWING ALLOCATIONS ARE FOR RUNNING UNDER MVS
-* DYNAM ALLOC FILE CURRCODE DA USER1.FOCEXEC.DATA(CURRCODE) SHR REU
-* DYNAM ALLOC FILE CURRDATA DA USER1.FOCEXEC.DATA(CURRDATA) SHR REU

SET EUROFILE = CURRCODE

DEFINE FILE CURRDATA
PRICEEUR/P17.2 CURR EUR = EUR(PRICE, ACTUAL);
END

TABLE FILE CURRDATA
PRINT PRICE PRICEEUR AND COMPUTE
PRICEFRF/P17.2 CURR FRF = FRF(PRICE, ACTUAL);
PRICEBEF/P17.2 CURR BEF = BEF(PRICE, ACTUAL);
END
```

This request generates the following report:

PAGE 1

PRICE	PRICEEUR	PRICEFRF	PRICEBEF
5.00	4.26	27.97	172.01
6.00	5.12	33.57	206.42
40.00	34.12	223.78	1376.20
10.00	8.53	55.95	344.06

Note: You cannot use the derived euro value PRICEEUR in a conversion from USD to BEF. PRICEEUR has two decimal places (P17.2), not three, as the triangulation rules require. Therefore, PRICEEUR yields the following inaccurate result (see PRICEBEF above) and is not valid as the intermediate value in a currency conversion that requires triangulation:

```
COMPUTE PRICENEW/P17.2 CURR BEF = BEF(PRICEEUR, ACTUAL);
```

```
PRICENEW  
-----  
171.85  
206.54  
1376.40  
344.10
```


NF744: HOLD FORMAT EXCEL

FOCUS can now format report output as a Microsoft® Excel spreadsheet. When you use the HOLD FORMAT EXCEL syntax, FOCUS creates a binary file containing all columns of the report output with their column headings.

If you are using the Web Interface for FOCUS, the Excel spreadsheet can be downloaded automatically to your Web Browser. If you do not have the Web Interface, you can transfer the spreadsheet file to any environment that supports Excel, such as a PC running Microsoft Windows®; the recommended transfer protocol is FTP in binary mode.

You can also store the Excel file on a web server where users can display it using any browser configured with Excel as a plug-in.

Syntax How to Create an Excel Spreadsheet

The syntax is.

```
[ON TABLE] HOLD [AS filename] FORMAT EXCEL
```

where:

`ON TABLE`

Is required syntax if you create the Excel file within a report request.

filename

Assigns a name to the Hold file. The default name is HOLD.

In MVS, unless you allocate this ddname to a permanent file, FOCUS allocates it to a temporary data set.

In CMS, this name becomes the file name. The file type is XLS.

By default, the extension for Microsoft Excel files is .xls in environments that support file names with extensions. Therefore, the file name after transfer is filename.xls.

Syntax How to Create and Download an Excel File With the Web Interface for FOCUS

If you are using the Web Interface for FOCUS, you can create the Excel spreadsheet and automatically download it to your Web Browser with the following syntax:

```
[ON TABLE] PCHOLD FORMAT EXCEL
```

where:

```
ON TABLE
```

Is required syntax if you create the Excel file within a report request.

On the PC, the name of the Excel spreadsheet file is *hold.xls*.

For a detailed description of how to download report output using the Web Interface for FOCUS and PCHOLD, see [NF730: Hold Format PDF](#).

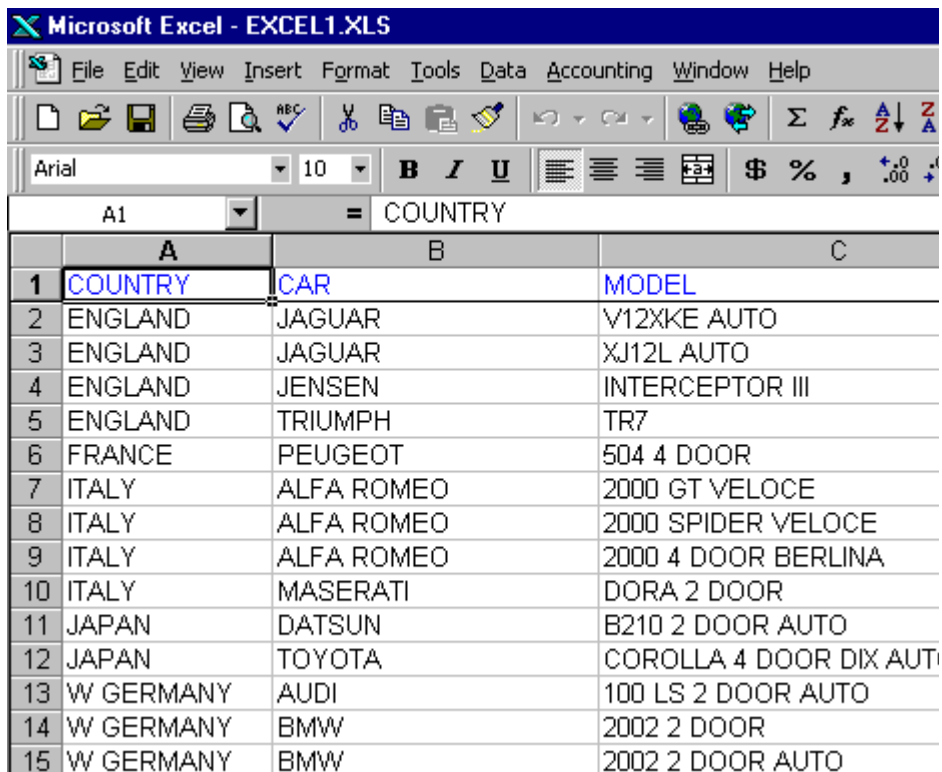
Example Creating an Excel Spreadsheet in a Report Request

Consider the following request run in CMS:

```
TABLE FILE CAR  
PRINT CAR MODEL RETAIL_COST DEALER_COST  
BY COUNTRY  
ON TABLE HOLD AS EXCEL1 FORMAT EXCEL  
END
```

NF744: HOLD FORMAT EXCEL

This request creates the file EXCEL1 XLS A. After transferring this file (using FTP in binary mode) to a file named *excel1.xls* on your PC, you can open it in Excel:



Microsoft Excel - EXCEL1.XLS

File Edit View Insert Format Tools Data Accounting Window Help

Arial 10 B I U \$ % , .00

A1 = COUNTRY

	A	B	C
1	COUNTRY	CAR	MODEL
2	ENGLAND	JAGUAR	V12XKE AUTO
3	ENGLAND	JAGUAR	XJ12L AUTO
4	ENGLAND	JENSEN	INTERCEPTOR III
5	ENGLAND	TRIUMPH	TR7
6	FRANCE	PEUGEOT	504 4 DOOR
7	ITALY	ALFA ROMEO	2000 GT VELOCE
8	ITALY	ALFA ROMEO	2000 SPIDER VELOCE
9	ITALY	ALFA ROMEO	2000 4 DOOR BERLINA
10	ITALY	MASERATI	DORA 2 DOOR
11	JAPAN	DATSUN	B210 2 DOOR AUTO
12	JAPAN	TOYOTA	COROLLA 4 DOOR DIX AUTO
13	W GERMANY	AUDI	100 LS 2 DOOR AUTO
14	W GERMANY	BMW	2002 2 DOOR
15	W GERMANY	BMW	2002 2 DOOR AUTO

NF730: Hold Format PDF

FOCUS can now generate output in Adobe® Portable Document Format (PDF). This feature enables Web Interface users to produce reports with all PDF formatting options (for example, headings, footings, and titles) correctly aligned on the physical pages. PDF also supports StyleSheets incorporating drill-downs and links to URLs.

Required Software Configuration

Adobe Acrobat Reader® Version 3.01 or higher is required to display PDF output, however, no third-party products are needed to *produce* it. The Adobe Acrobat Reader is Internet shareware for Windows 95, Windows NT, UNIX, and Macintosh, and is available free from their Web site, <http://www.adobe.com>.

Reports viewed with Adobe Acrobat Reader look precisely as if printed. By configuring the Acrobat Reader as a plug-in on your Web browser, PDF output displays directly inside the browser window in printed format without additional setup or configuration.

Browser users who have not installed Acrobat Reader can save PDF files to disk or download them to a PC when prompted by the browser and then transfer them later to a machine with the Acrobat Reader to display them. Users can then either run the standalone Acrobat Reader program or use a browser with the PDF plug-in to view the PDF files.

Syntax How to Create PDF Output

The syntax is:

```
ON TABLE {HOLD|SAVE} FORMAT PDF [AS filename]
```

where:

ON TABLE

Is required syntax if you create the PDF file within a report request.

filename

Assigns a name to the Hold file. The default name is HOLD. If you specify an optional AS filename the name that you supply (1-8 characters) is used in place of HOLD. If you issue a FILEDEF for the filename, the PDF output is created in the file specified in the FILEDEF.

In MVS, unless you allocate this ddname to a permanent file, FOCUS allocates it to a temporary dataset.

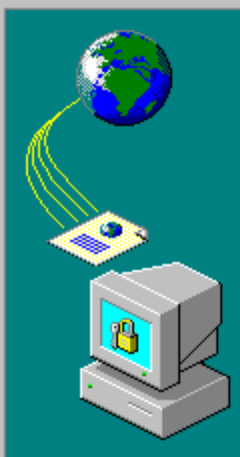
In CMS, this name becomes the filename. The filetype is PDF.

Downloading PDF Output

The technique for downloading PDF files depends on how you accessed FOCUS. PDF-formatted Hold files created on the mainframe can be downloaded to the PC using FTP transfers. With the Web Interface, browser users can issue the following PCHOLD syntax to download PDF files to their browsers.

ON TABLE PCHOLD FORMAT PDF

The Web Interface automatically FTPs output to your PC if you choose. A browser window opens after the command is issued, inquiring whether you wish to have the file opened immediately or saved to disk.



You have chosen to download a file from this location.

webtoe916f4b18. from ibivm

What would you like to do with this file?

Open this file from its current location

Save this file to disk

Always ask before opening this type of file

OK

Cancel

More Info

Example Creating a PDF-formatted Report Using the CAR File

This report request generates the PDF report that follows.

```
TABLE FILE CAR
HEADING CENTER
"CAR COSTS PER COUNTRY"
FOOTING
"----> Uncentered footing"
PRINT
MODEL DEALER_COST RETAIL_COST
BY COUNTRY
BY CAR
WHERE COUNTRY = 'ENGLAND' OR 'ITALY'
ON CAR SUBTOTAL
ON TABLE PCHOLD FORMAT PDF
END
```

Acrobat Reader - [HOLD.PDF]

PAGE 1

<u>COUNTRY</u>	<u>CAR</u>	<u>CAR COSTS PER COUNTRY</u> <u>MODEL</u>	<u>DEALER_COST</u>	<u>RETAIL_COST</u>
ENGLAND	JAGUAR	V12XKE AUTO	7,427	8,878
		XJ12L AUTO	11,194	13,491
*TOTAL CAR JAGUAR			18,621	22,369
	JENSEN	INTERCEPTOR III	14,940	17,850
*TOTAL CAR JENSEN			14,940	17,850
	TRIUMPH	TR7	4,292	5,100
*TOTAL CAR TRIUMPH			4,292	5,100
ITALY	ALFA ROMEO	2000 GT VELOCE	5,660	6,820
		2000 SPIDER VELOCE	5,660	6,820
		2000 4 DOOR BERLINA	4,915	5,925
*TOTAL CAR ALFA ROMEO			16,235	19,565
	MASERATI	DORA 2 DOOR	25,000	31,500
*TOTAL CAR MASERATI			25,000	31,500
TOTAL			79,088	96,384
--> UNCENTERED FOOTING				

Page 1 of 1 | 100% | 8.5 x 11 in

NF654: HOLD From External Sort

External sorts can be used to create HOLD files. This can lead to savings of up to twenty percent on processing time. The gains are most notable with relatively simple requests against large databases.

Syntax How to Create HOLD FILES with an External Sort

```
SET EXTHOLD = [OFF|ON]
```

where:

OFF

Disables HOLD files by an external sort. OFF is the default.

ON

Enables HOLD files by an external sort.

Conditions for Using External Sort to Create a HOLD File

- The default setting of EXTSORT=ON must be in effect.
- EXTHOLD must be ON.
- Request must contain a BY field.
- Request must contain ON TABLE HOLD or ON TABLE HOLD AS.
- Your query should be simple (AUTOTABLEFable). AUTOTABLEF analyzes a query and determines whether the combination of verbs and formatting options require the internal matrix or not. In cases where it's determined that a matrix is not necessary to satisfy the query we avoid the extra internal costs associated with creating the matrix. The internal matrix is stored in a file or dataset named FOCSORT. Its default is ON so that performance gains may be realized.
- SET ALL must be OFF.

NF654: HOLD From External Sort

- There cannot be an IF/WHERE TOTAL or BY TOTAL in the request.
- If a request contains a SUM command, EXTAGGR must be set ON and the only column prefixes allowed are SUM and FST.
- If a request contains a PRINT command, the column prefixes allowed are SUM, AVE, MAX, MIN, FST and LST.

NF597: Aggregation by External Sort

External sorts can be used to perform aggregation with a significant decrease in processing time in comparison to using the sort facility of FOCUS. The gains will be most notable with relatively simple requests against large databases.

Syntax How to Use Aggregation in Your External Sort

`SET EXTAGGR = aggropt`

where:

aggropt

can be one of the following:

`OFF`

disallows aggregation by an external sort.

`NOFLOAT`

allows aggregation if there are no floating data fields present.

`ON`

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

Conditions for Aggregating with an External Sort

- You must be using SYNC SORT or DFSORT.
- EXTAGGR cannot be set to OFF.
- Your query should be simple. (AUTOTABLE Fable)
- The PRINT display command may not be used in the query.
- SET ALL must be equal to OFF.
- Only the following column prefixes are allowed: SUM, AVG, CNT, FST.
- Columns can be COMPUTED or have a ROW-TOTAL.

- CMS DFSORT does not support aggregation of numeric data types. When SET EXTAGGR = NOFLOAT and your query aggregates numeric data, the external sort is not called and aggregation is performed by the FOCUS sort.

Example How Using an External Sort for Aggregation Can Change Your Output

Using an external sort for aggregation can change the output of your report request.

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

With aggregation in the external sort turned on:

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

The output is:

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

With aggregation in the external sort turned off:

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

The output is:

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

Note: The SET SUMPREFIX command in conjunction with aggregation using an external sort also affects the order of information displayed in your report. For complete information on SUMPREFIX please see New Feature Bulletin 728, SUMPREFIX.

Reference Special Considerations

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

NF728: Changing Retrieval Order with Aggregation

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

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

The SUMPREFIX command deals with this difference in behavior by allowing users to choose which order the answer set should display.

Syntax Setting Retrieval Order

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

where:

LST

Displays the last value in cases of data aggregation of alphanumeric or smart date data types.

FST

Displays the first value in cases of data aggregation of alphanumeric or smart date data types.

NF655: FOCPROF - The System Wide Profile

FOCPROF is a new global profile for FOCUS. While the FOCPARM profile supports only FOCUS SET commands, this new profile can contain any command that is valid in a FOCEXEC, including TABLE, GRAPH, MATCH, MODIFY, MAINTAIN, REBUILD, COMPILE, LOAD, Dialogue Manager commands, CMS commands, TSO commands, and DYNAM commands.

FOCUS Profiles

With the addition of FOCPROF, there are now three FOCUS profiles: FOCPARM, FOCPROF, and PROFILE.

The files FOCPARM and FOCPROF:

- Are members of the ERRORS PDS on MVS.
- Have filetype ERRORS on CMS.

The file PROFILE:

- Is a member of the FOCEXEC PDS on MVS.
- Has filetype FOCEXEC on CMS.

Using FOCUS Profiles

The order of execution of FOCUS profiles is:

1. FOCPARM, which can contain FOCUS SET commands only.
2. FOCPROF, the new global profile.
3. PROFILE.

Procedure How to Create a FOCPROF Profile

- For MVS, create a new member of the ERRORS PDS named FOCPROF.
- For CMS, create a file with filename FOCPROF and filetype ERRORS.

Edit the FOCPROF file to contain the commands to be executed each time FOCUS is invoked.

NF660: Multi-volume Support in MVS FOCUS

The latest release of MVS FOCUS gives sites the option of allocating FOCUS databases, Fusion databases, and FOCUS-created sequential files across multiple volumes.

Advantages of Multi-volume Data Sources

Many sites prefer to distribute high volume data sources across multiple volumes in order to manage:

- Use of storage on specific devices or device types.
- Run-time access to these devices.
- B37 abends.

You can now use this performance tuning technique (also known as *data striping*) with FOCUS databases, Fusion databases, and FOCUS-created sequential files in the MVS batch, TSO, and MSO environments.

Allocating Multi-volume Data Sources

The SPACE parameter for allocating a data source can include a primary and a secondary allocation. The primary allocation is the amount of space allocated the first time data is written to the data set. The secondary allocation is the amount of space to be allocated, when necessary, for up to 15 additional extents.

For a single-volume data source, processing terminates with a B37 abend when the system detects either of the following conditions:

- A need for more than 16 extents.
- A need for a new secondary extent (below the 16-extent limit) when enough space is not available on the volume.

FOCUS returns the following message to indicate that one of these conditions has occurred:

`(FOC198) FATAL ERROR IN DATABASE I/O. FOCUS TERMINATING CODE: 0000070`

You can prevent this type of abnormal termination by allocating multiple volumes to the data source. With multiple volumes, an out of space condition on the first volume causes allocation to start on another volume.

With multiple volumes, the allocation process varies slightly for each of the following:

- The first volume.
- Intermediate volumes.
- The last volume.

The following table describes the multi-volume allocation process:

Primary Allocation	The primary allocation is applied to the first volume only. It can consist of the number of extents allowed by MVS for a primary allocation. Note: A data source with no secondary space allocation is limited to a single volume.
--------------------	--

Secondary
Allocation

1.First volume:

As many extents as are available, up to the 16-extent limit, are allocated and filled before continuing to the second volume.

2.Intermediate volume:

Depending on the space available, up to 16 extents are filled before allocation begins on the next volume.

3.Last volume:

Once the need for a number of extents greater than the limit is detected:

- For a FOCUS or Fusion database, processing terminates with the following message
`(FOC198) FATAL ERROR IN DATABASE I/O. FOCUS
TERMINATING CODE 00000070`
- For temporary FOCSORT files, after all volumes and extents are filled allocation spills to up to 15 additional temporary files, each with 16 extents. (This feature, NF536, *Multi-Image FOCSORT*, was added in FOCUS 7.0.6.) The SPACE allocation for each spill file is the same as the SPACE allocation for the original FOCSORT file.

For example:

```
//FOCSORT DD SPACE=(TRK,(5,5))
```

This allocates a total of $5 + (5 * 15) = 80$ tracks. When the 81st track is needed, another temporary data set is allocated with the parameter `SPACE=(TRK,(5,5))`. If necessary, this additional step is repeated a total of 15 times yielding a total of $80 * 16$ tracks for FOCSORT.

If enough space is not available after filling all of the extents of all of the spill files, the FOC198 message is issued and processing terminates.

Note: The number of extents actually obtained on any volume may be less than 16; however, in most situations 16 will be available and used.

Syntax How to Allocate a Multi-volume Data Source in the MVS Batch Environment

You have two choices for statically allocating a new multi-volume FOCUS database, Fusion database, or sequential file.

You can list multiple VOLSER identifiers on the DD card for the multi-volume data source:

```
//ddname DD DSN=dsname,VOL=SER=(vol1,...,voln),...
```

Alternatively, you can ask for multiple units of a specific type:

```
//ddname DD DSN=dsname,UNIT=(type,n),...
```

where:

ddname

Is the DDNAME associated with the multi-volume data source.

dsname

Is the data set name of the multi-volume data source.

vol1,...,voln

Are the volume identifiers for the each of the volumes to use.

type

Is the type of unit to use.

n

Is the number of units.

Allocating a Multi-volume Data Source in the TSO and MSO Environments

In both TSO and MSO you have two choices for dynamically allocating a multi-volume data source:

- You can list multiple volume identifiers.
- You can specify the number of units to use and let the system choose the specific volumes. All of the units will be the same type (for example, 3390).

Syntax How to Allocate Specific Volumes in the TSO and MSO Environments

To allocate specific volumes for a multi-volume data source, use the following syntax:

- In TSO

```
TSO ALLOC ... VOLUME('vol1,...,voln')...
```

- In MVS FOCUS or MSO

```
DYNAM ALLOC ... VOL vol1,...,voln ...
```

where:

vol1,...,voln

Are the volume identifiers for the each of the volumes to use.

Syntax How to Specify the Number of Units in the TSO and MSO Environments

To specify the number of volumes for a multi-volume data source and let the system choose the specific volumes, use the following syntax:

- In TSO

```
TSO ALLOC ... UCOUNT('n') UNIT('type') ...
```

- In MVS FOCUS or MSO

```
DYNAM ALLOC ... UCOUNT n UNIT type ...
```

where:

n

Is the number of volumes to use.

type

Is the type of unit to use.

Note:

- UNIT VIO is not supported.
- The RLSE option of the SPACE parameter is not supported.
- The DYNAM UCOUNT parameter is also discussed in NF670, *DYNAM Support for Unit Count*.

Example Allocating a Data Source to Two Volumes

The following DYNAM command allocates two volumes to a data source called MULTVOL:

```
DYNAM ALLOC FI MULTVOL DS USER1.FOCTST.MULTVOL TRACK SPACE 4 4 REU -  
UCOUNT 2 UNIT SYSDA CATALOG
```

With this allocation, a second volume will be used when the 17th extent is needed.

Syntax How to Display the Volume Identifiers Allocated to a Multi-volume Data Source

To see the data set information associated with a specific DDNAME, issue the following command

```
? TSO DDNAME ddname
```

where:

ddname

Is the DDNAME allocated to the data set whose volume identifiers you want to see.

Example Displaying Multi-volume Data Set Information

The following example shows how to display data set information for DDNAME MULTVOL:

```
? tso ddname multvol
```

The following information is returned. Notice that two volume serial identifiers are listed on the VOLSER line:

```
DDNAME      = MULTVOL
DSNAME      = USER1.FOCTST.MULTVOL
DISP        = NEW
DEVICE      = DISK
VOLSER      = MFOC02,MFOC01
DSORG       = PS
RECFM       = F
SECONDARY   = 4
ALLOCATION   = TRACKS
BLKSIZE     = 4096
LRECL       = 4096
TRKTOT      = 92
EXTENTSUSED = 23
BLKSPERTRK = 12
TRKSPERCYL = 15
CYLSPERDISK = 2227
BLKSWRITTEN = 1104
FOCUSPAGES = 1059
> >
```

Choosing Default Sizes for FOCUS-created Files

IBITABLA, the file that contains default allocations for FOCUS-created files, has been enhanced to allow a unit count for FOCUS databases, Fusion databases, and all sequential FOCUS-created files. The advantages of multi-volume allocations are described in [Advantages of Multi-volume Data Sources](#).

Every FOCUS release is shipped with a member of the FOCCTL.DATA PDS named IBITABLA. This member contains default allocations for all FOCUS-created files that you do not specifically allocate in your FOCUS session, JCL, or CLIST. As part of the installation process, the FOCUS installer should copy the new version of IBITABLA to the ERRORS PDS and edit it to conform to the standards for the site.

When a new release of FOCUS is shipped, the installer should compare the latest version of IBITABLA to the prior site-specific version in order to construct a new site-specific version of IBITABLA. In this way, the installer will be aware of changes in format (such as new fields or ddnames added in the new release) that must be addressed in the customized copy.

The following is the default IBITABLA shipped with FOCUS Version 7.0 Release 9:

```
* . . . + . . . 1 . . . + . . . 2 . . . + . . . 3 . . . + . . . 4 . . . +
HOLD      CYLS    5  10                                3,
HOLDMAST TRKS    5  5 36                                ,
SAVE      CYLS    5  10                                3,
REBUILD  CYLS    5  10                                3,
FOCSML   CYLS    5  5                                  2,
FOCUS    CYLS    5  5                                  1,
FOCSTACK TRKS    5  5                                  2,
FOCSORT  CYLS    5  5                                  1,
OFFLINE  CYLS                                A           ,
SESSION  TRKS    5  5                                  2,
FOCCOMP  TRKS    5  5 12                                ,
HOLDACC  TRKS    5  5 12                                ,
FMU      TRKS    5  5 12                                ,
TRF      TRKS    5  5 12                                ,
FOCPOOLT CYLS    5  20                                NOHIPER 4,
FUSION   CYLS    5  50                                NOHIPER 4,
MDI     CYLS    5  20                                NOHIPER 4,
FOC$HOLD CYLS    5  5                                  2,
EXTINDEX CYLS    5  5                                  2,
```

The unit count field is columns 44-45. The fields are:

Columns	Length	Comments
01-08	8	Class of file - DDname
10-13	4	Allocation units (CYLS, TRKS)
15-17	3	Primary allocation
19-21	3	Secondary allocation
23-24	2	Number of directory blocks (blank specifies a sequential file; 0 is invalid)
26-26	1	SYSOUT class
28-33	6	Volume serial on which to allocate
35-42	8	Type of unit to allocate (for example, 3390, DASD, NOHIPER*)
44-45	2	Unit count

Note that partitioned data sets do not support multi-volume allocations.

If you enter a non-valid value for any field in IBITABLA, FOCUS substitutes the corresponding value from the original version of IBITABLA that was shipped with FOCUS.

Example Allocating DDNAME FOCSTACK to Two Volumes in IBITABLA

Assume you replaced the original FOCSTACK line with the following in IBITABLA:

```
|...+...1...+...2...+...3...+...4...+  
FOCSTACK TRKS 5 5 3390 2,
```

This line indicates the following allocation attributes:

- DDname = FOCSTACK
- Allocation is in tracks:
Primary tracks=5
Secondary tracks=5
- Type of unit = 3390
- Number of units=2

The resulting dynamic allocation is equivalent to the following JCL:

```
//FOCSTACK DD SPACE=(TRK,(5,5)),UNIT=(3390,2),...
```

Reference Usage Notes for IBITABLA

- IBITABLA is a fixed-format file.
- Each data field must be placed in specific columns, but leading or trailing blanks are allowed. (There is at least one blank between successive data fields.)
- All lines beginning with an asterisk (*) are comments.
- Unit count is ignored for HiperFOCUS files with DISP=(NEW,DELETE).

NF584: Dynamically Setting the IDMS DBNAME and DICTNAME

The new IDMS Interface commands SET DBNAME and SET DICTNAME enable you to dynamically change the DBNAME and DICTNAME parameters at any time during your FOCUS session.

If you do not issue these commands, the Interface reads the DBNAME and DICTNAME from the Access File. However, once you issue them, the new DBNAME and DICTNAME values take precedence over those in your Access Files. The new values remain in effect until you either:

- Reissue the SET commands with new DBNAME and DICTNAME values.
- End your FOCUS session.
- Reinstate the Access File parameters by issuing the SET commands with the DEFAULT option.

Syntax How to Set the DBNAME and DICTNAME

Issue the following commands at the FOCUS session prompt, in a FOCEXEC, or in any supported profile:

```
TSO IDMSR SET DBNAME {dbname|DEFAULT}
```

```
TSO IDMSR SET DICTNAME {dictname|DEFAULT}
```

where:

dbname

Is the IDMS database name that you want to access.

dictname

Is the IDMS dictionary name that you want to access.

DEFAULT

Causes the Interface to read the value from the Access File.

Syntax **How to Display the Current Settings**

To display the settings that are currently in effect, issue the following command:

```
TSO IDMSR SET ?
```

Example **Dynamically Changing the DBNAME and DICTNAME Values**

```
TSO IDMSR SET DBNAME EMPDEMO
```

```
TSO IDMSR SET DICTNAME APPLDICT
```

NF673: Model 204 Interface Account Split

The Model 204 Interface has been enhanced to accept an account code as part of the logon string.

The syntax for specifying the logon account in the Access File is

```
ACCOUNT=x[ y], ACCOUNTPASS=pswd, IFAMCHNL=ifchnl, $
```

The SET command syntax for specifying the logon account is

```
{TSO|MVS} M204IN SET ACCTNAME x[ y]  
{TSO|MVS} M204IN SET ACCTPASS pswd
```

where:

TSO|MVS

Is a required environmental prefix. TSO and MVS are synonyms and can be used interchangeably.

x

Is the user ID, up to 16 characters.

y

Is the account code, up to 15 characters, preceded by exactly one blank.

pswd

Is the account password.

Example Setting the Model 204 Userid and Account Code

Assume the Model 204 userid is SUPERKLUGE and the account code is MIS.

You can issue the following Interface command to set these values:

```
TSO M204IN SET ACCTNAME SUPERKLUGE MIS
```

You can also set them in the Access File with the following attribute:

```
ACCOUNT=SUPERKLUGE MIS, ...
```

NF720: SQLJOIN OUTER Setting for Relational Interfaces

With the new SQLJOIN OUTER setting you can control when the relational Interfaces optimize outer joins without affecting the optimization of other operations. This parameter provides backward compatibility with prior releases of the relational Interfaces and enables you to fine-tune your applications.

When join optimization is in effect, the Interface generates one SQL SELECT statement that includes every table involved in the join. The RDBMS can then process the join. When join optimization is disabled, the Interface generates a separate SQL SELECT statement for each table, and FOCUS processes the join.

In FOCUS Version 7.0 Release 6, the relational Interfaces were enhanced to optimize outer joins. The following command causes the Interface to generate an outer join:

```
SET ALL = ON
```

Beginning with FOCUS Version 7.0 Release 6, issuing this command with OPTIMIZATION enabled invokes outer join optimization. In FOCUS releases prior to 7.0.6, this setting disabled optimization so that FOCUS always processed the outer join. Turning OPTIMIZATION OFF still causes FOCUS to process the join, but it disables *all* optimization enhancements, not just outer join processing.

Starting with FOCUS Version 7.0 Release 9, you can use the SQLJOIN OUTER setting to disable outer join optimization while leaving other optimization enhancements in effect.

Syntax How to Control Outer Join Optimization

Issue the following command at the FOCUS prompt, in a stored procedure, or in any supported profile:

```
SQL target_db SET SQLJOIN OUTER {ON|OFF}
```

where:

`target_db`

Indicates the target RDBMS. Valid values are DB2, SQLDS, SQLDBC, SQLORA, or SQLIDMS. Omit if you issued the SET SQLENGINE command.

`ON`

Enables outer join optimization.

`OFF`

Disables outer join optimization. OFF is the default value.

Note:

- The SQLJOIN OUTER setting is available only when optimization is enabled (that is, OPTIMIZATION is not set to OFF).
- The SQLJOIN OUTER setting is ignored when SET ALL = OFF.

Effects of Combinations of Settings on Outer Join Optimization

The following table describes how different combinations of OPTIMIZATION and SQLJOIN OUTER settings affect Interface behavior. It assumes that SET ALL = ON:

Settings		Results	
OPTIMIZATION	SQLJOIN OUTER	Outer Join Optimized?	Other Optimization Features
ON	ON	Yes	Enabled
ON	OFF	No	Enabled
OFF	N/A	No	Disabled

Settings		Results	
OPTIMIZATION	SQLJOIN OUTER	Outer Join Optimized?	Other Optimization Features
SQL	ON	Yes, in all possible cases	Enabled
SQL	OFF	No	Enabled
FOCUS	ON	Yes if results are equivalent to FOCUS managed request	Enabled
FOCUS	OFF	No	Enabled

Reference SQLJOIN OUTER Messages

If SQLJOIN OUTER is set to OFF, the following message displays when you issue the SQL ? query command:

```
(FOC1420) OPTIMIZATION OF ALL=ON AS LEFT JOIN - : OFF
```

NF652: Teradata Interface Kanji Support

The Teradata Interface now supports the DBCS (Double Byte Character Set) Kanji characters described by the Teradata datatypes GRAPHIC and VARGRAPHIC.

The following conversion chart shows the SQL datatypes that support the Kanji character set and their corresponding ACTUAL formats in the Master File:

SQL Datatype	ACTUAL Format	Description
GRAPHIC	Kn	DBCS Kanji character set. Fixed-length string of 'n' 16-bit characters where $0 < n \leq 127$. The appropriate corresponding USAGE format is A(2n+2).
VARGRAPHIC	Kn	DBCS Kanji character set. Varying-length string of 'n' 16-bit characters where $0 < n \leq 127$. The appropriate corresponding USAGE format is A(2n+2). Note: LONG VARGRAPHIC (or VARGRAPHIC (n) where $n > 127$) is not supported.

NF722: FOCUS Client DNS Names Support

This feature enables the FOCUS Client EDACFG file to identify an EDA server by host name rather than IP address. The EDACFG file is the client communications configuration file allocated to DDNAME EDACFG. In prior versions of FOCUS Client, the EDACFG file had to supply the IP address of the EDA server to which it would connect.

The domain name system (DNS) is a global network of servers that translate host names, such as `www.ibi.com`, into IP addresses. For more information about FOCUS Client, see NF494, *FOCUS Client - Remote Data Access via EDA/SQL*, which was implemented in FOCUS 7.0.5.

Syntax How to Invoke DNS Names Support

The syntax for specifying a host name in the client configuration file for TCP/IP is

```
HOST = hostname
```

where:

```
hostname
```

Is the host name of the EDA server.

Example Using DNS Names Support

The following client configuration file is used for connecting to the host named IBIMVS:

```
NAME = EDA/SQL CLIENT USING CS/3 TCP/IP
NODE = TCPOUT
  BEGIN
; TRACE = 31
  PROTOCOL = TCP
  CLASS = CLIENT
  HOST = IBIMVS ; DNS NAME OF HOST
  SERVICE = 2459 ; PORT NUMBER OF THE EDA SERVER
  END
```

NF656: Controlling REBUILD Messages

This feature allows for direct control over the frequency with which REBUILD issues messages. FOCUS, by default, displays a message for every 1000 records read during the database retrieval and load phases of the REBUILD utility. The message, REFERENCE..AT SEGMENT 1000, REFERENCE..AT SEGMENT 2000.. is a function of the number of records in the FOCUS file being rebuilt. The frequency of these messages can become problematic for larger FOCUS files because CMS spool space may be limited.

Syntax How to Control REBUILD Messages

The user can set how often the message is displayed by issuing the command:

```
SET REBUILDMSG = n
```

where:

n

is any 8-byte integer.

A setting of less than 1000 generates a diagnostic and keeps the current setting. The current setting will either be the default of 1000, or the last valid integer greater than 999 to which REBUILDMSG was set. A setting of 0 disables the 'REFERENCE..AT SEGMENT' messages.

Example Controlling Display of REBUILD Messages

The following example shows a REBUILD CHECK function where REBUILDMSG has been set to 4000, and the database contains 19,753 records.

```
ENTER NAME OF FOCUS/FUSION FILE (FN FT FM)
...
STARTING..
REFERENCE..AT SEGMENT      4000
REFERENCE..AT SEGMENT      8000
REFERENCE..AT SEGMENT     12000
REFERENCE..AT SEGMENT     16000
NUMBER OF SEGMENTS RETRIEVED=  19753
CHECK COMPLETED...
```

NF670: DYNAM Support for Unit Count

The DYNAM command now supports the unit count parameter for allocating multi-volume data sources. With the UCOUNT parameter you can allocate a file to multiple volumes of a particular type of storage device without indicating specific volume serial numbers.

Advantages of Multi-volume Data Sources

Many sites prefer to distribute high volume data sources across multiple volumes in order to manage:

- Use of storage on specific devices or device types.
- Run-time access to these devices.

This new DYNAM parameter is particularly useful for allocating large temporary files such as FOCSORT, FOCPOOLT, and HOLD files in the TSO and MSO environments.

Syntax How to Specify the DYNAM Unit Count Parameter

```
DYNAM ALLOC ... UCOUNT n UNIT type ...
```

where:

n

Is the number of volumes to use.

type

Is the type of unit to use. **Note:** UNIT VIO is not supported.

Example Allocating a HOLD File to Two Volumes

The following DYNAM command allocates two 3390s to a HOLD file:

```
DYNAM ALLOC FILE HOLD SPACE 20,10 TRACKS UCOUNT 2 UNIT 3390
```

NF684: PCHOLD for Non-Html Files

Web Interface support of PCHOLD enables Web browser users to extract five types of preformatted data from the mainframe and either display the output immediately on their browsers or automatically transfer the files via FTP to their PCs.

Using PCHOLD for Formats LOTUS, DIF, EXCEL, or PDF

When you issue the PCHOLD command for files with LOTUS, DIF, EXCEL, or PDF formats, the Web Interface returns a notification window telling you that the file is being downloaded. You can then rename the file on the PC using the correct extension from the table below. Word processing files (Format=WP) are always displayed first on the browser and can then be saved to the PC.

Syntax Specifying PCHOLD in a TABLE Request

`ON TABLE PCHOLD FORMAT fmt`

where:

fmt

Specifies the format of the PCHOLD extract file. When downloading these files to the PC you must use the appropriate extensions from the following table:

File Format	Content	Extension
LOTUS	Lotus PRN files	.prn
DIF	Spreadsheets without headings	.dif
EXCEL	Excel spreadsheets with headings	.xls
PDF	ADOBE Portable Document Format (PDF) files	.pdf
WP	Word processing files	.txt or .doc

Syntax Specifying PCHOLD for a WP File in a TABLE Request

`ON TABLE PCHOLD FORMAT WP`

WP-formatted files are displayed immediately on the browser and can be saved to the PC. When the file appears, select the Save As option, providing a file name with an appropriate file extension for your word processor (for example, extension for MS Word = .doc).

`SAVE AS myfile.doc`

Example Using PCHOLD for Non-Html Files

The request below creates a spreadsheet without headings (format=DIF) displayed on the screen that follows:

```
table file car
print compute sales1/i9 = sales;
profit/d10.2 = retail_cost - dealer_cost;
adate/yymd = '1999/04/29';
by model
ON TABLE PCHOLD FORMAT DIF
end
```

Microsoft Excel - testdif

File Edit View Insert Format Tools Data Window Help

Arial 10 B I U \$ % , +.00 -.00 100%

	A	B	C	D	E	F	G
1	B210 2 DOOR AUTO	43000	513	19990429			
2	COROLLA 4 DOOR DIX AUTO	35030	453	19990429			
3	DORA 2 DOOR	0	6500	19990429			
4	INTERCEPTOR III	0	2910	19990429			
5	TR7	0	808	19990429			
6	V12XKE AUTO	0	1451	19990429			
7	XJ12L AUTO	12000	2297	19990429			
8	100 LS 2 DOOR AUTO	7800	907	19990429			
9	2000 GT VELOCE	12400	1160	19990429			
10	2000 SPIDER VELOCE	13000	1160	19990429			
11	2000 4 DOOR BERLINA	4800	1010	19990429			
12	2002 2 DOOR	8950	140	19990429			
13	2002 2 DOOR AUTO	8900	355	19990429			
14	3.0 SI 4 DOOR	14000	3752	19990429			
15	3.0 SI 4 DOOR AUTO	18940	3123	19990429			
16	504 4 DOOR	0	979	19990429			
17	530I 4 DOOR	14000	797	19990429			
18	530I 4 DOOR AUTO	15600	1095	19990429			

Reference Notes for Internet Explorer Users

- **LOTUS or DIF.** When using Internet Explorer to view consecutive LOTUS or DIF reports online you must take special precautions to insure getting latest version of temporary files.

This step must be taken to avoid a potential problem caused by the use of the same filename when running consecutive reports for immediate viewing (as opposed to saving the output in uniquely named files). Spreadsheet users planning to view multiple consecutive reports online must configure the Internet Explorer browser in the following manner:

- **Internet Explorer Release 5** - Under Tools/Internet Options/Temporary Internet Files, select Check for newer versions automatically.
- **Internet Explorer Release 4** - Under View options/General/Temporary Internet Files, select Check every time.

This step is not required with Netscape Navigator, which checks temporary Internet files for newer versions every time as the default. Users of other browsers must check their browser's handling of this situation and act accordingly.

- **Excel.** When using Internet Explorer to view Excel output you must use the binary MIME type for the file transfer (all other file types are transferred as text).

NF683: Web Interface support for Maintain Winforms

This feature enables mainframe FOCUS sites to deploy applications created with the Cactus Workbench to intranet or Internet audiences without installing an EDA or Cactus server on the host machine.

Existing applications can be “webified” by using the Cactus Workbench to convert Maintain Winforms into Webforms. The Focexecs (.fex files) and Winforms (.wfm files) must then be transmitted **manually** to the mainframe using FTP (alphanumeric mode):

- For MVS, transfer the FOCEXEC files to a data set allocated to ddname FOCEXEC and the Winform files to a data set allocated to ddname WINFORMS in your FOCUS CLIST.
- For CMS, transfer the files to your A disk. The FOCEXEC files have filetype FOCEXEC and the Winform files have filetype WINFORMS.

Once these are installed, end users running the Web Interface from a Web browser can execute these applications and access data defined to mainframe FOCUS.

Prerequisites

Web390 Server Release 3.2 is required to use this feature and end-users will need to install at least Release 4 of Netscape Navigator, Microsoft Internet Explorer, or an equivalently featured browser.

Procedure How to Execute Maintain Applications From a Browser

There are two ways to execute Maintain applications from the browser window:

1. Use Logon scripts (*Web390 Developer's Guide and Installation Manual* DN1001035.0599 provides instructions for creating Logon scripts).

Logon scripts can bring browser users directly into the Webform equivalent of a Maintain Winform after they enter the appropriate userid and password. Thereafter, they can work interactively with the applications on their browsers.

2. Users can execute applications directly from the Web Interface Interactive screen

EX FOCEXECname

where:

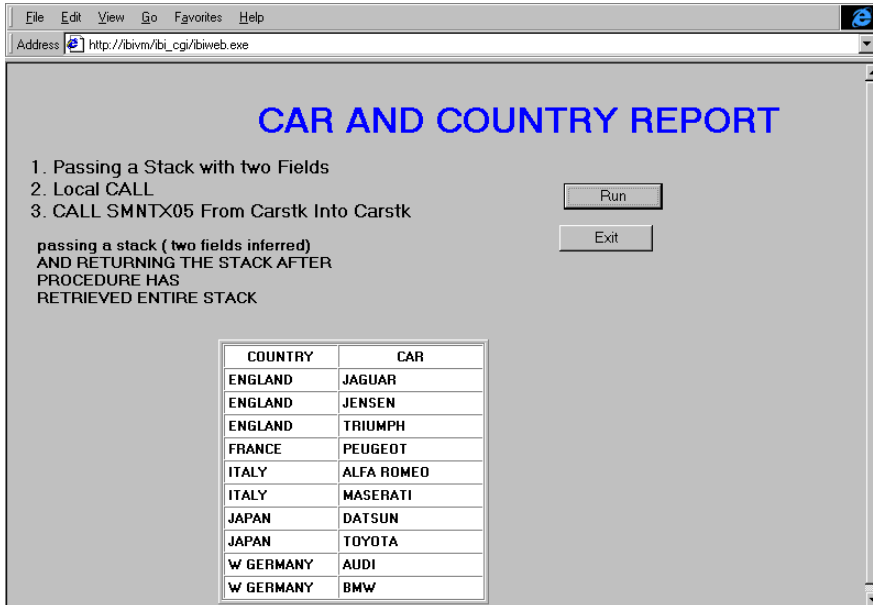
FOCEXECname

Is the name of the target Maintain application.

Example Running a Maintain Application From a Browser

This webified Maintain application issues a simple car-by-country report request, formatting the output on the Webform below:

EX CMNTX05W



The screenshot shows a web browser window with the address bar containing `http://ibivm/ibi_cgi/ibiweb.exe`. The main content area displays the following:

CAR AND COUNTRY REPORT

1. Passing a Stack with two Fields
2. Local CALL
3. CALL SMNTX05 From Carstk Into Carstk

passing a stack (two fields inferred)
AND RETURNING THE STACK AFTER
PROCEDURE HAS
RETRIEVED ENTIRE STACK

Run

Exit

COUNTRY	CAR
ENGLAND	JAGUAR
ENGLAND	JENSEN
ENGLAND	TRIUMPH
FRANCE	PEUGEOT
ITALY	ALFA ROMEO
ITALY	MASERATI
JAPAN	DATSUN
JAPAN	TOYOTA
W GERMANY	AUDI
W GERMANY	BMW

NF691: Escape Character for LIKE Predicate

New syntax exists for the LIKE predicate that enables you to search for special characters in data. You can set an escape character to use in the LIKE predicate. Then, when you include this escape character in front of a special character in the mask pattern, the parser treats the special character as a normal character and searches for it in the data. This next character can be '%' or '_' which are normally special characters. LIKE may be coded in WHERE, DEFINE, and COMPUTE statements. The mask is an alphanumeric, user-supplied pattern that FOCUS uses to compare characters in a data field value. A mask has two special characters:

1. The percent sign (%) to indicate any number of characters; and
2. The underscore (_) to indicate a single character in a specified position.

The escape character enhancement permits FOCUS to treat these masking characters as literals within the search pattern and not as wildcards.

Any single character can be used as an escape character, but the one used must be prefaced with the word ESCAPE. The syntax is:

```
WHERE field LIKE 'phrase' ESCAPE 'c'
```

where

```
'c'
```

Is any character embedded in the phrase before a '%' or '_'.

Escape Character Capabilities

The escape character is supported by the SQL Interfaces, as well as FOCUS databases. The escape character is only in effect when the ESCAPE syntax is included in the LIKE predicate.

Syntax How to Use the Escape Character

The pattern (alphanumeric constant) that follows the LIKE predicate may contain wildcard characters ('%') and ('_') that users may need to escape in order to use as part of the search pattern. Every LIKE predicate can provide its own escape character to be used within the pattern (or mask). The syntax is as follows:

```
WHERE expression LIKE 'abc\%' ESCAPE '\'
```

where

expression

Is the name of the expression to be evaluated in the selection test.

'abc\%'

Is the alphanumeric test value.

'\'

Is the escape character enclosed in single quotation marks. When the escape character is used in the pattern immediately preceding the special character '%' or '_', FOCUS is instructed to treat the special character as a literal and not as a wildcard. The character itself can also be escaped, thus becoming a normal character in a string (for example, 'abc\%\%').

Example Using the Escape Character

Using the Car file, assume that both Peugeot and Alfa Romeo produce 4_door car models. To generate a report showing countries that produce 4_door car models, the syntax would be as follows:

```
TABLE FILE CAR  
PRINT CAR MODEL  
BY COUNTRY  
WHERE MODEL LIKE '%g_DOOR%' ESCAPE 'g'  
END
```

The request produces the following report:

PAGE 1

COUNTRY	CAR	MODEL
-----	---	-----
FRANCE	PEUGEOT	504 4_DOOR
ITALY	ALFA ROMEO	2000 4_DOOR BERLINA

Reference Special Considerations

- The use of an escape character in front of any character other than '%', '_', and itself will be ignored.
- Only one escape character can be used per LIKE phrase.
- If a WHERE clause is used with lazy ORs, the ESCAPE must be on the first phrase and will apply to all subsequent phrases in that WHERE clause. For example:

```
WHERE field LIKE 'ABCg_' ESCAPE 'g' OR 'ABCg%' OR 'g%ABC'
```

Reference Error Messages

(FOC36251) SYNTAX ERROR IN LIKE OPERATOR

Alphanumeric mask must follow the LIKE operator. An optional keyword ESCAPE followed by a single character alphanumeric constant in apostrophes can be used after the mask.

NF718: DYNAM Support for Existing Relative GDG Numbers

The DYNAM command now supports allocation of existing iterations of a Generation Data Group (GDG) using relative index numbers. Existing iterations are those with index numbers less than or equal to zero.

Note: DYNAM does *not* support allocation of a new GDG iteration; that is, you cannot use the relative number +1 in a DYNAM allocation for a GDG.

Using DYNAM With Relative GDG Numbers

The original definition of a GDG assigns it a group name and specifies how many generations will be maintained. Once the maximum number of generations has been reached, each new iteration replaces the oldest existing iteration. The data set name for each iteration is the group name appended with a qualifier that contains the iteration number.

For example, the following GDG, named USER1.HOLD.GDG, has three generations:

Data Set Name	Iteration Number	Index Number
USER1.HOLD.GDG.G0003V00	3 (Oldest)	-2
USER1.HOLD.GDG.G0004V00	4	-1
USER1.HOLD.GDG.G0005V00	5 (Current)	0

The current (that is, newest) iteration always has the highest iteration number and corresponds to index number zero. The next newest corresponds to index number -1, and so on.

Prior to this new feature, DYNAM could allocate only the current iteration of a GDG. Now DYNAM supports relative index numbers for allocating any existing iteration.

Syntax How to Use DYNAM With Relative Index Numbers

Use the following syntax to allocate an existing iteration of a GDG

```
DYNAM ALLOC FILE ddname DS 'group_name(index)' SHR REUSE
```

where:

ddname

Is the name of the Master File for the GDG.

group_name

Is the group name of the GDG.

index

Is the index number for an existing iteration of the GDG. It must be less than or equal to zero.

Example Allocating Existing GDG Iterations Using DYNAM With Relative Index Numbers

The GDG named USER1.HOLD.GDG discussed in [Using DYNAM With Relative GDG Numbers](#), is described by Master File GDG1:

```
FILENAME=GDG1 , SUFFIX=FIX  
SEGNAME=ORIGIN , SEGTYPE=S1  
FIELDNAME=FLD1 , , A10 , $  
FIELDNAME=FLD2 , , A20 , $  
FIELDNAME=FLD3 , , A30 , $  
FIELDNAME=FLD4 , , A10 , $
```

In each iteration, the value stored in FLD1 is the generation number. For example, USER1.HOLD.GDG.G0004V00 contains the following records:

```
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
4444444444444AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

The following request allocates and prints FLD1 of iteration four (index number -1)

```
> dynam alloc file gdgl ds 'user1.hold.gdg(-1)' shr reu
> > table file gdgl
> print fld1
> end
PAGE      1
```

```
FLD1
----
4444444444
4444444444
4444444444
4444444444
4444444444
```

The following DYNAM command allocates the current iteration of the GDG, and the TABLE request accesses this latest iteration:

```
> dynam free file gdgl
> > dynam alloc file gdgl ds 'user1.hold.gdg(0)' shr reu
> > table file gdgl
> print fld1
> where recordlimit eq 1
> end
```

PAGE 1

FLD1

5555555555

Syntax Determining Which GDG Iteration is Allocated

To determine which GDG iteration is allocated, issue the ? TSO DDNAME query command:

```
> > ? tso ddname gdg1
```

```
DDNAME      = GDG1
DSNAME      = USER1.HOLD.GDG.G0003V00
DISP        = SHR
DEVICE      = DISK
VOLSER      = USERME
DSORG       = PS
RECFM       = FB
SECONDARY   = 5
ALLOCATION   = TRACKS
BLKSIZE     = 8000
LRECL       = 80
TRKTOT      = 1
EXTENTSUSED = 1
BLKSPERTRK = 6
TRKSPERCYL = 15
CYLSPERDISK = 3340
BLKSWRITTEN = 1
> >
```

The DSNAME, USER1.HOLD.GDG.G0003V00, indicates that the third iteration is allocated.

Reference Error Messages

The MESSAGE parameter must be set ON in order to receive these messages.

An attempt to allocate a new iteration of a GDG (index number +1) generates the following error message:

```
(FOC880)      DYNAMIC ALLOCATION ERROR: IKJ56871I DATA SET name
              NOT ALLOCATED, RELATIVE GENERATION NUMBER INCOMPATIBLE FOR
              SPECIFIED STATUS, #099:0C-0394
```

Allocating an index number that is out of range (too large a negative number), generates the following message:

```
(FOC855)      UNABLE TO LOCATE DATASET name, #099:04-1708
```

NF735: Enhancement to ? SET

Two options have been added to the ? SET command. They are ? SET FOR and ? SET NOT. The FOR option lists the current state of the command queried, and details where it may be set within FOCUS. The NOT option produces a list of SET commands not settable in five specific areas.

Syntax Querying a Command

```
? SET FOR parameter
```

where:

```
parameter
```

Is any SET parameter.

Example Querying the Current State of EXTSORT

Entering

```
? SET FOR EXTSORT
```

yields

```
EXTSORT                                ON
```

```
-----  
SETTABLE FROM COMMAND LINE             : YES  
SETTABLE ON TABLE                     : YES  
SETTABLE FROM SYSTEM-WIDE PROFILE      : YES  
SETTABLE FROM HLI PROFILE              : YES  
POOL TABLE BOUNDARY                  : YES
```

```
>
```

The preceding screen shows that EXTSORT is currently set ON and that it is settable from all five features.

Syntax **Determining Which Commands Are Not Settable In Each of the Five Features**

? SET NOT *functional_area*

where:

functional_area

can be one of the following five areas:

- PROMPT (in a PROMPT command)
- ONTABLE (in a report request)
- FOCPARM (in the FOCPARM profile)
- HLIPROF (in the HLI profile)
- PT (Pooled Tables)

Example Determining Which Commands Are Not Settable In a Report Request

Entering

? SET NOT ONTABLE

yields:

NON-SETTABLE ON TABLE PARAMETER SETTINGS

BINS	64	LANGUAGE	AMENGLISH	REBUILDMSG	1000
BLKCALC	NEW	MAXPOOLMEM	32768	SAVEMATRIX	ON
BYPANELING	OFF	MDIBINS	8000	TCPIPINT	OFF
CACHE	0	MDIPROGRESS	100000	TEMP DISK	A
COLUMNSCROLL	OFF	MODE	CMS	TRMSD	24
DATEDISPLAY	OFF	MPRINT	NEW	TRMSW	80
DATEFNS	ON	POOL	OFF	TRMTYP	1 (3270)
DEFCENT	19	POOLBATCH	OFF	WEBHOME	OFF
EUROFILE		POOLFEATURE	OFF	WIDTH	130
FIELDNAME	NEW	POOLMEMORY	16384	WINPFKEY	OLD
FOCSTACK SIZE	8	POOLRESERVE	1024	YRTHRESH	0
HTMLMODE	OFF	PRINTPLUS	OFF		

>

The preceding screen shows a list of parameters that are not settable using ON TABLE.

NF740: Changes to the REBUILD Prompt

With the addition of two new options, MIGRATE and MDINDEX, the REBUILD prompt no longer fits on one line. After you enter REBUILD at the FOCUS prompt a numbered list is displayed that allows you to enter either the option name or its corresponding number.

Syntax The REBUILD Prompt Screen

```
>REBUILD
```

```
Enter option
```

1. REBUILD (Optimize the database structure)
2. REORG (Alter the database structure)
3. INDEX (Build/modify the database index)
4. EXTERNAL INDEX (Build/modify an external index database)
5. CHECK (Check the database structure)
6. TIMESTAMP (Change the database timestamp)
7. DATE NEW (Convert old date formats to smartdate formats)
8. MDINDEX (Build/modify a multidimensional index. FUSION DBs only)
9. MIGRATE (Convert FOCUS masters/DBs to FUSION)

Note: A change was also made to the FILENAME prompt. Instead of prompting for the name of the FOCUS file, you are now asked for the FOCUS/FUSION filename. Within the individual options a similar numbered list is also displayed if applicable. REBUILD commands are stackable in name or number format.

NF745: ? PTF Enhancements

Two columns have been added to the output displayed in response to ? PTF. They are, SUPERSEDED BY, which displays a superseding PTF if one exists and PUT LEVEL, which demonstrates the order in which multiple PTFs must be applied..

Syntax How to Identify PTFs Applied to Your Version of FOCUS

Issue the following command at the FOCUS prompt:

```
? PTF
```

A screen similar to the following is displayed:

```
>
? ptf
PTFS APPLIED TO RELEASE 70XFOC
FROM PTFTABLE LOCATED IN IBITEST LOADLIB C1

COUNT PTF NUM    CREATED    APPLIED    SUPERSEDED BY    PUT LEVEL
-----
1) 95828          .....    .....    112600          .....
2) 107164         .....    .....    112600          .....
3) 110763         .....    .....    112600          .....
4) 112600         19990427  19990513  .....          200295
>
```

Note: Dots denote that no information exists for a column entry in the resulting report.

NF746: Leading Zeros

This feature may be used in Dialogue Manager for date subroutines that return a numeric integer format. It specifically addresses the case where a two-digit year is input in a Dialogue Manager string. The result of the subroutine is 00, representing the year 2000. The leading zeros are truncated when typed out.

Syntax Displaying Leading Zeros

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

where:

ON

Allows the display of leading zeros if they are present.

OFF

Truncates leading zeros if they are present.

Example Preserving Leading Zeros

In the following example the AYM subroutine is being called. The input year is 99 and the month is 12.

```
-SET &IN = '9912';  
-SET &OUT = AYM (&IN, 1, 'I4');  
-TYPE &OUT
```

This yields

1

Adding

```
SET LEADZERO = ON
```

before the above example yields

0001

correctly indicating January. 2000.

Note: LEADZERO only supports expressions that make a direct call to a subroutine. Expressions that have nesting or other mathematical functions truncate leading zeros. For example,

```
-SET &OUT = AYM(&IN, 1, 'I4'/100;
```

NF748: HOLD FORMAT WP With Carriage Control

You can now control whether carriage control characters appear in column 1 of each page of report output saved with the HOLD FORMAT WP option. When you choose to include carriage control in a format WP file in MVS, FOCUS creates the file with RECFM VBA so that each report page prints on a separate sheet of paper.

Syntax How to Include Carriage Control Characters in a FORMAT WP Hold File

```
[ON TABLE] HOLD AS filename FORMAT WP [CC|NOCC]
```

where:

filename

Is the name of the resulting file.

CC

Includes a carriage control character (1) as the first character of each report page in the extract file. In MVS, the file is created with RECFM VBA which causes the operating system to respect the carriage control via page ejects.

NOCC

Omits carriage control characters from the extract file. The file is created with RECFM VB.

When you do not include either CC or NOCC in the syntax, the value of the PAGE-NUM (or PAGE) parameter controls whether carriage control characters are supplied in column 1:

- SET PAGE-NUM=ON (the default) or SET PAGE-NUM=NOPAGE causes column 1 to be blank (no carriage control). In MVS, the file is created with RECFM=VB.

- SET PAGE-NUM=OFF, SET PAGE-NUM=TOP, or using the TABPAGENO option in the heading of the report request supplies carriage control characters. This is the same as prior behavior. However, starting with this release, these settings cause the Hold file to be created with RECFM VBA in MVS.

Reference Directing Output to a Printer

In MVS, you can send the format WP output directly to a printer by allocating the filename to that printer. You must FREE or CLOSE the file after it is created and before it is printed. For example:

```
SET PAGE = ON
  DYNAM ALLOC FILE EMPLOYEE DA PREFIX.EMPLOYEE.FOCUS SHR REU
  DYNAM FREE FILE HOLD1
  DYNAM ALLOC FILE HOLD1 SYSOUT A DEST IBIVM.P29C1 CLOSE REUSE
  TABLE FILE EMPLOYEE
  PRINT EMP_ID
  BY CURR_SAL BY LAST_NAME
  ON TABLE HOLD AS HOLD1 FORMAT WP CC
END
```

Note: Your SYSOUT allocation must contain your destination printer.

In VM/CMS, print the HOLD file using the CC option of the CMS PRINT command to interpret the first character of each record as a carriage control character.

```
PRINT fn ft fm (cc
```

Reminder: If the HOLD file is created with carriage control, it must be printed with carriage control.

Reference **FORMAT WP Carriage Control and MVS Record Format Options**

The following table summarizes the options for creating a FORMAT WP extract file:

Carriage Control Option	PAGE=ON or NOPAGE		PAGE=OFF or TOP or TABPAGENO in heading	
	Carriage Control?	MVS RECFM	Carriage Control?	MVS RECFM
CC	Yes	VBA	Yes	VBA
NOCC	No	VB	No	VB
none	No	VB	Yes	VBA

7.0.8R New Features

Year 2000

FOCUS

[NF557: REBUILD - Legacy Date Conversion](#)

[NF653: Displaying Base Dates in FOCUS Reports](#)

[NF659: CHECK FILE HOLD ALL](#)

[NF700: New Date Math Functions for the Year 2000](#)

[NF703: Displaying Invalid Smart Dates in Reports](#)

[NF705: Enhancement to the YRTHRESH Command](#)

[NF708: Enhancement to the TODAY Subroutine](#)

[NF709: Displaying a Date Variable Without Separators](#)

[NF710: Field FORMAT=YYJUL](#)

[NF711: Altering Your System Date for Testing Purposes](#)

[NF713: MSO Log Changes](#)

General Enhancements

[NF714: LE Support](#)

NF557: REBUILD - Legacy Date Conversion

A new option has been added to REBUILD for legacy date conversion. The option DATE NEW converts legacy dates to smart dates in your FOCUS databases. The utility uses 'update-in-place' technology. It updates your database and creates a new Master File, yet does not change the structure or size of the database.

You must backup the database before executing REBUILD with the DATE NEW option. We recommend that you run the utility against the copy and then replace the original file with the updated backup.

How the REBUILD Utility Converts Legacy Dates

The utility overwrites the original legacy date field with a smart date. When the storage size of the legacy date exceeds four bytes (the storage size of a smart date), a pad field is added to the database following the date field. Formats A6YMD, A6MDY, and A6DMY are changed to formats YMD, MDY, and DMY, respectively, and have a two-byte pad field added to the Master File. The storage size of integer dates (I6YMD, I6MDY, for example) is four bytes, so no pad field is added. All packed fields and A8 dates add a four-byte pad field. When a date is a key field (but not the last key for the segment), and it requires a pad field, the number of keys in the SEGTYPE is increased by one for each date field that requires padding.

The utility only changes legacy dates to smart dates. The FORMAT in the Master File must be one of the following (Month translation edit options T and TR may be included in the format):

A8YYMD A8MDYY A8DMYY A6YMD A6MDY A6DMY A6YYM A6MYM A4YM A4MY
I8YYMD I8MDYY I8DMYY I6YMD I6MDY I6DMY I6YYM I6MYM I4YM I4MY
P8YYMD P8MDYY P8DMYY P6YMD P6MDY P6DMY P6YYM P6MYM P4YM P4MY

If you have a field that stores date values but does not have one of these formats, the utility does not change it. If you have a date with one of these formats that you do not want changed, temporarily remove the date components from the FORMAT, run the REBUILD, and then restore the date format.

Example Using REBUILD With the DATE NEW Option

```
let clear *
set defcent = 19
set yrthresh = 50
set pass = dohide
use

employee focus f
end
rebuild
ENTER OPTION (REBUILD,REORG,INDEX,EXTERNAL INDEX,CHECK OR TIMESTAMP,DATE
NEW)
date new
ENTER THE NAME OF THE MASTER
employee
ENTER THE NEW NAME FOR THE MASTER (FN FT FM)
newemp master a
HAVE YOU BACKED UP THE DATABASE? (YES,NO)
yes
```

If you answer anything other than YES to 'HAVE YOU BACKED UP THE DATABASE', the REBUILD does not continue. Backing up your database is critical to this process.

In CMS, the new Master File is written to the file that you specified in the prompt. The default filetype is MASTER, and the default filemode is A.

In MVS, the new Master File is written to *ddname* HOLDMAST. After the new Master File is created, you should immediately copy it to a permanent dataset.

For example:

```
DYNAM COPYDD HOLDMAST(newname) MASTER(newname)
```

Notes:

- The DBA password for the file must be issued prior to issuing REBUILD.
- The original Master File cannot be encrypted.
- All files must be available locally during the REBUILD, including LOCATION files.
- The Master File cannot have GROUP fields.
- Some error numbers are available in &FOCERRNUM while all error numbers are available in &&FOCREBUILD. Test both &&FOCREBUILD and &FOCERRNUM for errors when writing procedures to rebuild your files.
- To avoid any potential problems, clear all LETs and JOINS before issuing REBUILD.
- DEFCENT/YRTHRESH are respected at the global, file, and field level.
- Correct all invalid date values in the database before executing REBUILD/DATE NEW. The utility converts all invalid dates to zero. Invalid dates used as keys may lead to duplicate keys in the database.
- Adequate workspace, such as temporary attached disk storage, must be available for the temporary REBUILD file. As a rule of thumb, have space 10 to 20% larger than the size of the existing file available.
- REBUILD/INDEX is performed automatically if an index exists.
- REBUILD/REBUILD is performed automatically after REBUILD/DATE NEW when any key is a date.

- Sort libraries and workspace must be available (as with REBUILD/INDEX). The REBUILD allocates default sortwork space in MVS, if you have not already. DDNAMEs SORTIN and SORTOUT must be allocated prior to issuing a REBUILD:

```
DYNAM ALLOC FILE SORTIN   SPACE 5,5 TRACKS NEW
DYNAM ALLOC FILE SORTOUT  SPACE 5,5 TRACKS NEW
DYNAM ALLOC FILE SYSOUT DA * MOD
```

Restrictions

REBUILD DATE NEW is a remediation tool for your FOCUS databases and date fields only. It does not remediate:

- DEFINES in the Master File
- ACCEPTS in the Master File
- DBA restrictions (especially VALUE restrictions) in the Master File or DBAFILE
- Cross-references to other date fields in this or other Master Files.
- Any references to date fields in your application (FOCEXECs, Master Files).

Updated Master File Created by REBUILD/DATE NEW

The REBUILD/DATE NEW utility creates an updated Master File that reflects the changes made to the database. Once the file has been rebuilt, the original Master File can no longer be used against the database. You must use the new Master File created by the REBUILD/DATE NEW utility.

For example:

```
USE
EMPLOYEE {FOCUS F} AS NEWEMP
END
REBUILD
DATE NEW
EMPLOYEE
NEWEMP
YES
-RUN
-IF (&&FOCREBUILD NE 0)OR(&FOCERRNUM NE 0) GOTO error_case;
USE
EMPLOYEE {FOCUS F} AS NEWEMP
END
TABLE FILE NEWEMP
PRINT ....
END
```

The new Master File is an updated copy of the original Master File except:

- The USAGE format for legacy date fields is updated to remove the format and length. The date edit options are retained. For example A6YMDTR becomes YMDTR.
- Padding fields are added for those dates that need them:

```
FIELDNAME= ,ALIAS= ,FORMAT=An,$ PAD FIELD ADDED BY REBUILD
```

where *n* is the padding length (either 2 or 4). Note that the FIELDNAME and ALIAS are blank.

- The SEGTYPE attribute is updated for segments that have remediated dates as keys when the date requires padding and the date is not the last field in the key. The SEGTYPE number will be increased by the number of pad fields added to the key.
- If the SEGTYPE is missing for any segment, the following line is added immediately prior to the \$ terminator for that segment:

```
SEGTYPE=segtype,$ OMITTED SEGTYPE ADDED BY REBUILD
```

where *segtype* is determined by FOCUS.

- If the USAGE for any field, including date fields is missing, the following line is added, immediately prior to the \$ terminator for that field:

```
USAGE=fmt,$ OMITTED USAGE ADDED BY REBUILD
```

where *fmt* is the format of the previous field in the Master File. FOCUS automatically assigns the previous field's format to any field coded without an explicit USAGE= statement.

Example Sample Master File: Before and After Conversion

Before Conversion	After Conversion
FILE=filename	FILE=newfilename
SEGNAME=segname, SEGTYPE=S2	SEGNAME=segname, SEGTYPE=S3
FIELD=KEY1,,USAGE=A6YMD,\$	FIELD=KEY1,,USAGE= YMD,\$
	FIELD=, ,USAGE=A2,\$ PAD FIELD ADDED BY REBUILD
FIELD=KEY2,,USAGE=I6MDY,\$	FIELD=KEY2,,USAGE= MDY,\$
FIELD=FIELD3,,USAGE=A8YYMD ,,\$	FIELD=FIELD3,,USAGE= YYMD,\$
	FIELD=, ,USAGE=A4,\$ PAD FIELD ADDED BY REBUILD

In the conversion of the Master File:

- The SEGTYPE changes from an S2 to S3 to incorporate a 2-byte pad field.
- Format A6YMD changes to smart date format YMD.

- A 2-byte pad field with a blank fieldname and alias is added to the Master File.
- Format I6MDY changes to smart date format MDY (no pad needed).
- Format A8YYMD changes to smart date format YYMD.
- A 4-byte pad field with a blank fieldname and alias is added to the Master File.

Action Taken on a Date Field During REBUILD/DATE NEW

A new message has been added after a REBUILD has been performed:

NUMBER OF SEGMENTS CHANGED= *n*

where:

n is the number of segments that have been changed. For example, if there are 10 fields on one segment, and 20 records, then *n* is 20 (the number of records/segments changed).

REBUILD/DATE NEW performs a REBUILD/REBUILD or REBUILD/INDEX automatically when a date field is a key or a date field is indexed. The following chart shows the action taken on a date field during the REBUILD/DATE NEW process.

Date is a Key	Index	Result
No	None	NUMBER OF SEGMENTS CHANGED = <i>n</i>
No	Yes	REBUILD/INDEX on date field
Yes	None	REBUILD/REBUILD is performed.
Yes	On Any field	REBUILD/REBUILD is performed. REBUILD/INDEX is performed for the indexed fields.

REBUILD/DATE NEW Error Messages

40001 - THIS UTILITY IS NOT SUPPORTED ON THIS PLATFORM
40002 - FILE NOT BACKED UP - REBUILD NOT EXECUTED
40003 - THERE ARE NO DATE FIELDS IN THE FILE - REBUILD NOT EXECUTED
40004 - FILE CONTAINS GROUP FIELDS - REBUILD NOT EXECUTED
40005 - COMBINE FILE CANNOT BE REBUILT
40006 - INTERNAL ERROR IN REBUILD/DATE NEW
40007 - NEW MASTER NAME MUST BE DIFFERENT THAN THE ORIGINAL

NF653: Displaying Base Dates in FOCUS Reports

You can now display base dates in a FOCUS report. Previously, TABLE always displayed a blank when:

- A date read from a file matched the base date, or
- A field with a smart date format had the value 0

The following chart shows the base date for each supported date format:

FORMAT	Base Date
YMD and YYMD	1900/12/31
YM and YYM	1901/01
YQ and YYQ	1901/Q1
JUL and YYJUL	00/365 and 1900/365 respectively

Syntax Invoking DATEDISPLAY

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

where:

ON

Displays the base date if the data is the base date value.

OFF

Displays blanks if the data is the base date value. OFF is the default.

You cannot set DATEDISPLAY with the ON TABLE command.

NF659: CHECK FILE HOLD ALL

CHECK FILE HOLD has been enhanced so you can view all of the attributes in a HOLD file including YRTHRESH and DEFCENT. The HOLD file contains two new columns with the values of FDFCENT and FYRTHRESH at the file level and two new columns with the values of DEFCENT and YRTHRESH at the field level.

The syntax is:

```
CHECK FILE filename HOLD ALL
```

where

filename

Is the name of the file whose Master specifications are to be placed in a HOLD file

Then issue the following to see the data about the Master:

```
TABLE FILE HOLD  
PRINT *  
END
```

Running this report request displays columns FDFCENT and FYRTHRESH at the file level,

FDFCENT	FYRTHRESH
-----	-----
19	50
19	50

and columns DEFCENT and YRTHRESH at the field level.

DEFCENT	YRTHRESH
-----	-----
19	60
19	50

The four columns shown in the previous two examples represent a small portion of the total information displayed by the TABLE FILE HOLD command.

NF700: New Date Math Functions for the Year 2000

Several functions have been added to FOCUS enabling you to perform operations on day-based **new dates** in DEFINEs, COMPUTE and anywhere else a function can be used.

New Date Function Capabilities

The new date functions can:

- Add or subtract date units (months, years, days, weekdays or business days) to or from a date
- Yield a difference between dates in date units
- Move a date to a specific point in the calendar, such as End-of-Month
- Convert from one date format to another (including old dates)

These new functions can help you:

- Compute payroll dates
- Track and ship orders
- Ensure correct credit card transactions

Non day-based date calculations (for example, YM, YQ) can be computed in direct operations (+, -), so they do not need these functions.

Syntax Adding and Subtracting Date Units to or from a Date

You can add or subtract date units to or from a date by issuing the following:

```
DATEADD (YYMDate, 'unit', #units)
```

where

YYMDate

Is any day-based new date, for example, YYMD, MDY, or JUL

unit

Can be one of the following:

- Y (Year)
- M (Month)
- D (Day)
- WD (Weekday)
- BD (Business Day)

#units

Is the number of date units you wish to add or subtract to or from the day-based new date

For example,

```
DATE/YYMD = '19991231';
```

```
NEWDATE/YYMD = DATEADD (DATE, 'D', 5);
```

adds 5 days to yield a value of 2000/01/05

The number of units passed to DATEADD is always a whole unit. For example,

```
DATEADD (date, 'M', 1.999)
```

adds 1 month because the number of units is less than 2. Any fractional part is ignored. If the number of units is negative, DATEADD performs subtraction instead of addition.

Invalid date units result in a zero being returned.

If the result of adding months creates an invalid day in the new month, the day is backed off to the end of the resultant month. For example, adding one month to March 31st cannot yield April 31st. Instead, it correctly yields April 30th. The same is true for adding one month to January 29th, 30th, or 31st. All three result in the last day of February (28th, or 29th if a leap year). DATEADD works with smart dates only.

The following uses DATEADD to determine whether a date is a business day:

```
SET EMPTYREPORT=ON
DEFINE FILE DATE
  X/YYMD=DATEADD(D1_YYMD, 'BD', 0);
END
TABLE FILE DATE
HEADING
" USE DATEADD TO DETERMINE WHETHER A SMARTDATE FIELD IS A BUSINESS      "
" DAY.  THE DATABASE HAS THE DATE '1998/06/05' WHICH IS A FRIDAY        "
" STORED IN FIELD D1_YYMD.  AN IF TEST IS USED TO DETERMINE IF THE      "
" DATE CORRESPONDS TO A BUSINESS DAY.                                    "
  PRINT D1_YYMD X
  IF X EQ '19980605'
END
TABLE FILE DATE
HEADING
" IT WILL YIELD 0 RECORDS 0 LINES IF THE RESULTING DATE IS NOT A        "
" A BUSINESS DAY.  THE DATABASE ALSO HAS '1998/06/06, A SATURDAY.        "
  PRINT D1_YYMD X
  IF X EQ '19980606'
END
```


The preceding FOCEXEC yields the following:

PAGE 1

USE DATEADD TO DETERMINE WHETHER A SMARTDATE FIELD IS A BUSINESS DAY. THE DATABASE HAS THE DATE '1998/06/05' WHICH IS A FRIDAY STORED IN FIELD D1_YYMD. AN IF TEST IS USED TO DETERMINE IF THE DATE CORRESPONDS TO A BUSINESS DAY.

```
D1_YYMD      X
-----      -
1998/06/05   1998/06/05
```

PAGE 1

IT WILL YIELD 0 RECORDS 0 LINES IF THE RESULTING DATE IS NOT A BUSINESS DAY. THE DATABASE ALSO HAS '1998/06/06, A SATURDAY.

```
D1_YYMD      X
-----      -
```

Weekday Units

Weekday units (WD), by default, refer to Monday through Friday. One weekday past a Friday is the following Monday. If the input to DATEADD using WD is a Saturday or Sunday, the input is adjusted to the next weekday before doing the addition. For example,

```
DATEADD (Saturday, 'WD', 1)
```

and

```
DATEADD (Sunday, 'WD', 1)
```

both yield Tuesday as a result because Saturday and Sunday are not business days, so DATEADD begins with Monday and adds 1, yielding Tuesday.

Business Day Units

You can direct which days are considered business days and which days are not. Business days are traditionally Monday through Friday, but not every business works the same schedule. For example, if your company does business on Sunday, Tuesday, Wednesday, Friday, and Saturday, you can tailor business day units to reflect that situation. Issue the following positionally dependent command to set business days:

```
SET BUSDAYS = {day-list|_MTWTF_}
```

where

`day-list`

is the list of days that represents your business week. The list has a position for each day from Sunday to Saturday.

`_MTWTF_`

Represents the positional days of the week from Sunday through Saturday. The underscores represent Sunday and Saturday respectively. Monday through Friday with an underscore before and an underscore afterwards is the default.

Any day that you do not wish to designate as a business day must be replaced with an underscore (`_`) in its designated space. If any position within SMTWTF`S` is either not in its correct position or is not an underscore, an error message is displayed. Using the example of a company that does business on Sunday, Tuesday, Wednesday, Friday and Saturday, business days are represented as:

`S_TW_FS`

To view the current setting of business days issue:

```
? SET ALL or ? SET
```

Holidays

You also have the ability to individually tailor holiday schedules that affect the calculation of business days by skipping those days when calculating offsets. For example, in a given week, if Friday is designated as a holiday, the next business day (BD) after Thursday is the following Monday. In MVS the list of holidays is loaded from a member in ERRORS called HDAYxxxx. In CMS the list is loaded from HDAYxxx ERRORS. A sample Master File, (HDAYDB), and FOCEXEC, (HDAYMAKE), that creates an errors member from a data source used to maintain a list of holidays is available on the FOCUS disk. Create a flat file of holidays as described in the FOCEXEC and execute the FOCEXEC to create the holiday file. The value of xxxx is controlled by the SET HDAY command so that a single installation can support different holiday schedules.

For example,

```
SET HDAY = STKM
```

Reads in the holidays from member HDAYSTKM. Each year for which data exists must be represented in the holiday file. Calling a date function with a date value outside the range of the holidays file returns a zero on BD requests. The current setting of HDAY can be viewed with

```
? SET ALL or ? SET
```

Syntax Returning the Difference between Two Dates

You can return the difference between two dates by issuing the following:

```
DATEDIF (fromYYMD, toYYMD, 'unit')
```

where

fromYYMD

Is the starting date from which to calculate the difference

toYYMD

Is the ending date from which to calculate the difference

unit

See *Adding and Subtracting Date Units to or from a Date* for valid units. The number of units returned from DATEDIF is always a whole number. For example,

```
DATEDIF (19960302,19970301,'Y')
```

DATEDIF calculates to zero because the difference between March 2, 1996 and March 1, 1997 is less than a whole year. If the to-date is before the from-date, a negative number is returned. For example,

```
DATEDIF (19990228, 19990128, 'M')
```

```
DATEDIF (19990228, 19990129, 'M')
```

```
DATEDIF (19990228, 19990130, 'M')
```

```
DATEDIF (19990228, 19990131, 'M')
```

all return a result of minus 1 month.

Using DATEDIF with month units yields the inverse of DATEADD. If adding one month to date X creates date Y, then the count of months via DATEDIF between date X and date Y must be one month. The rule is:

If the to-date is an end-of-month then the month difference may be rounded up (in absolute terms) to guarantee the inverse rule. For example,

```
DATEDIF (March31, May31, 'M') yields 2
```

```
DATEDIF (March31, May30, 'M') yields 1 (because May 30 is not the end of the month)
```

```
DATEDIF (March31, April30, 'M') yields 1
```

The same rules apply to year math, the only difference being that February 29th plus 1 year is equal to February 28th.

DATEDIF works with smart dates only.

Syntax Moving a Date to a Significant Point

You can move a date to a significant point on the calendar by issuing the following:

```
DATEMOV (YYMDate, 'move-point')
```

where

YYMDate

Is the date you wish to move. May be any new date format as long as it implies a day component (for example MDYY, DMY, but not YM or MYY).

move-point

Is the significant point to which you wish to move. Permissible move-points are:

- EOM End of month
- BOM Beginning of month
- EOQ End of quarter
- BOQ Beginning of quarter
- EOY End of year
- BOY Beginning of year
- EOW End of week
- BOW Beginning of week
- NWD Next weekday
- NBD Next business day (Affected by BUSDAY and HDAY files)
- PWD Prior weekday
- PBD Prior business day (Affected by BUSDAY and HDAY files)
- WD- A weekday or earlier
- BD- A business day or earlier (Affected by BUSDAY and HDAY files)

- WD+ A weekday or later
- BD+ A business day or later (Affected by BUSDAY and HDAY files)

DATEMOV works with smart dates only.

The following shows an application using DATEMOV and the report it produces

```
DEFINE FILE CAR
ANUM/I5 WITH COUNTRY = ANUM +1;
ADATEX/YMD WITH COUNTRY = 19980507;
ADATE/YMD = ADATEX + ANUM;
NWD/YMDWT = DATEMOV(ADATE, 'NWD' );
PWD/YMDWT = DATEMOV(ADATE, 'PWD' );
WDP/YMDWT = DATEMOV(ADATE, 'WD+' );
WDM/YMDWT = DATEMOV(ADATE, 'WD-' );
NBD/YMDWT = DATEMOV(ADATE, 'NBD' );
PBD/YMDWT = DATEMOV(ADATE, 'PBD' );
WBP/YMDWT = DATEMOV(ADATE, 'BD+' );
WBM/YMDWT = DATEMOV(ADATE, 'BD-' );
END
SET BUSDAY = _MTWT__
TABLE FILE CAR
HEADING
"EXAMPLES OF DATEMOV"
"BUSINESS DAYS ARE MONDAY, TUESDAY, WEDNESDAY, + THURSDAY "
" "
"START DATE.. 3 MOVE POINTS....."
PRINT ADATE/WT AS 'DOW'
NWD/WT PWD/WT WDP/WT AS 'WD+' WDM/WT AS 'WD-'
NBD/WT PBD/WT WBP/WT AS 'BD+' WBM/WT AS 'BD-'
BY ADATE
END
```

yields:

EXAMPLES OF DATEMOV

BUSINESS DAYS ARE MONDAY, TUESDAY, WEDNESDAY, + THURSDAY

START DATE..	MOVE POINTS.....								
ADATE	DOW	NWD	PWD	WD+	WD-	NBD	PBD	BD+	BD-
-----	---	---	---	---	---	---	---	---	---
98/05/08	FRI	MON	THU	FRI	FRI	TUE	WED	MON	THU
98/05/09	SAT	TUE	THU	MON	FRI	TUE	WED	MON	THU
98/05/10	SUN	TUE	THU	MON	FRI	TUE	WED	MON	THU
98/05/11	MON	TUE	FRI	MON	MON	TUE	THU	MON	MON
98/05/12	TUE	WED	MON	TUE	TUE	WED	MON	TUE	TUE

Invalid move-points result in a zero being returned.

Syntax Converting From One Date Format to Another

Applications no longer have to use intermediate calculations to convert date formats. Instead you can issue the following:

```
DATECVT ( indate, 'infmt', 'outfmt' )
```

where

indate

Is the date whose format you wish to change

infmt and *outfmt*

Can be:

- Any new date format (for example, YYMD, YQ, M, DMY, JUL) that matches the format of *indate*. It can also be in the format of the output value enclosed within single quotes.

- Any old date format (such as I6YMD or A8MDYY)
- Non-date formats (such as I8, or A6). Non-date formats on infmt are treated as offsets from the base date (12/31/1900). Use the DAYMD function to retrieve the offset of a date.

The format of the field on the left side of the equal sign must match the outfmt value. For example,

```
field/DMY = DATECVT (indate, 'YYMD', 'DMY');
```

If the value of indate is 19991231 then the field is set to the offset, which is 311299. Indates with old formats obey any DEFCEINT and YRTHRESH values implied for that field when performing the conversion.

Invalid old dates passed to DATECVT cause a zero to be returned as does a DEFINE. Invalid formats in DATECVT cause a zero or blank to be returned.

New Date Math Functions in MAINTAIN

MAINTAIN supports the new date functions DATEADD, DATEDIF, and DATEMOV with an extra parameter: the result field. In MAINTAIN, you can code:

```
COMPUTE ADATE/YYMD = ... (some expression)
COMPUTE DUE_DATE/YYMD = DATEADD(ADATE, 'BD', 20, DUE_DATE);
COMPUTE NOTICE_DATE/YYMD = DATEMOV(DUE_DATE+1, 'EOM', NOTICE_DATE);
COMPUTE TOTAL_DAY/I4 = DATEDIF(ADATE, NOTICE_DATE, 'BD', TOTAL_DAY);
```

DATECVT is not supported in MAINTAIN. If you attempt to use DATECVT in MAINTAIN the following message displays:

```
FUNCTION NOT FOUND ERROR
```


NF703: Displaying Invalid Smart Dates in Reports

In previous releases of FOCUS, if a date field in a non-FOCUS file contained an invalid date, a diagnostic error was displayed and the entire record failed to display in a report. For example, if a date field contained '980450' with an ACTUAL of A6 and a USAGE of YMD, the record containing that field would not display. With the use of a new command, it is possible to display the rest of the record that contains the incorrect date.

Syntax Invoking ALLOWCVTERR

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

where

ON

Allows the display of a field containing an incorrect date.

OFF

Generates an error if bad data is encountered, and does not display the record containing the bad data. This behavior is identical to releases prior to FOCUS Release 7.0.8R.

When it encounters a bad date, ALLOWCVTERR sets the value of the field to either MISSING or to the base date. When a field is being converted and ALLOWCVTERR is set on, FOCUS first checks to see if MISSING=ON. The following chart shows the results of interaction between DATEDISPLAY and MISSING assuming ALLOWCVTERR=ON and the presence of a bad date.

	MISSING=OFF	MISSING=ON
DATEDISPLAY=ON	Displays Base Date 19001231 or 1901/1	.
DATEDISPLAY=OFF	Displays Blanks	.

DATEDISPLAY only affects how the base date is displayed.

Note: See New Feature Bulletin 653, *Displaying Base Dates in FOCUS Reports*, for detailed information concerning the setting of DATEDISPLAY.

NF705: Enhancement to the YRTHRESH Command

You can now set YRTHRESH as an offset from the current year in addition to specifying a year. This technique creates a moving century window that increments itself each year without modifying your application.

You decide the number of years to offset in YRTHRESH. For example, if the current year is 1998 and you wish to set YRTHRESH to 60, you can set YRTHRESH to -38 ($1998 - 38 = 60$). By setting YRTHRESH to a negative number FOCUS subtracts, in this example, 38 from whatever the current year is. In the year 1999 YRTHRESH is 61 instead of 60 ($1999 - 38 = 61$) illustrating how the moving window application functions without outside intervention.

If you set YRTHRESH to a large enough value that crosses a century boundary, the value of DEFCENT is recalculated. For example, if you set YRTHRESH to minus 99 ($1998 - 99 = -1$), DEFCENT is calculated to 18 and YRTHRESH becomes 99. The 100- year span begins with a pivot year of 1899 and ends with year 1998. ? SET and ? SET ALL now reflect the new settings of DEFCENT.

The following request shows the use of an offset with DEFCENT set to 19 and YRTHRESH set to -38 (where the current year is 1998), followed by the output:

```
SET DEFCENT=19, YRTHRESH=-38
TABLE FILE DATE
PRINT D2_I6YMD AND COMPUTE
NEWDATE/I8YYMD=AYMD(D2_I6YMD,1,NEWDATE);
END
```

NF705: Enhancement to the YRTHRESH Command

D2_I6YMD	NEWDATE
-----	-----
60/04/02	1960/04/03
66/06/06	1966/06/07
60/12/13	1960/12/14
53/06/06	2053/06/07
59/08/11	2059/08/12
60/02/28	1960/02/29

NF708: Enhancement to the TODAY Subroutine

The TODAY subroutine is year 2000 compatible and is useful in a compiled MODIFY. It can return a 4-digit year when you declare a DEFINE or COMPUTE field as 10 bytes. TODAY can continue returning a 2-digit year when you declare the output format as 8 bytes.

For example,

```
DEFINE FILE EMPLOYEE
NOWDATE/A10 WITH EMP_ID = TODAY (NOWDATE)
END

TABLE FILE EMPLOYEE
HEADING
"SALARY REPORT RUN ON DATE <NOWDATE>"
" "
PRINT DEPARTMENT CURR_SAL
BY LAST_NAME BY FIRST_NAME
END
```

The DEFINE may also be coded as

```
NOWDATE/A10 WITH EMP_ID = TODAY('A10');
```

The request produces the following report:

SALARY REPORT RUN ON DATE 06/08/1998			
LAST_NAME	FIRST_NAME	DEPARTMENT	CURR_SAL
-----	-----	-----	-----
BANNING	JOHN	PRODUCTION	\$29,700.00
BLACKWOOD	ROSEMARIE	MIS	\$21,780.00
CROSS	BARBARA	MIS	\$27,062.00
GREENSPAN	MARY	MIS	\$9,000.00
IRVING	JOAN	PRODUCTION	\$26,862.00
JONES	DIANE	MIS	\$18,480.00
MCCOY	JOHN	MIS	\$18,480.00
MCKNIGHT	ROGER	PRODUCTION	\$16,100.00
ROMANS	ANTHONY	PRODUCTION	\$21,120.00
SMITH	MARY	MIS	\$13,200.00
	RICHARD	PRODUCTION	\$9,500.00
STEVENS	ALFRED	PRODUCTION	\$11,000.00

Note: DATEFNS must be set to ON to retrieve the extended TODAY value.

NF709: Displaying a Date Variable Without Separators

You can now display a date variable containing a 4-digit year without separators. The variables are &YYMD, &MDYY and &DMYY. These variables complement the existing 2-digit year variables &YMD, &MDY, and &DMY.

The following example shows a report using &YYMD:

```
TABLE FILE EMPLOYEE
HEADING
"SALARY REPORT RUN ON DATE &YYMD"
" "
PRINT DEPARTMENT CURR_SAL
BY LAST_NAME BY FIRST_NAME
END
```

The resulting output for May 18, 1998 is:

SALARY REPORT RUN ON DATE 19980518

LAST_NAME	FIRST_NAME	DEPARTMENT	CURR_SAL
-----	-----	-----	-----
BANNING	JOHN	PRODUCTION	\$29,700.00
BLACKWOOD	ROSEMARIE	MIS	\$21,780.00
CROSS	BARBARA	MIS	\$27,062.00
GREENSPAN	MARY	MIS	\$9,000.00
IRVING	JOAN	PRODUCTION	\$26,862.00
JONES	DIANE	MIS	\$18,480.00
MCCOY	JOHN	MIS	\$18,480.00
MCKNIGHT	ROGER	PRODUCTION	\$16,100.00
ROMANS	ANTHONY	PRODUCTION	\$21,120.00
SMITH	MARY	MIS	\$13,200.00
	RICHARD	PRODUCTION	\$9,500.00
STEVENS	ALFRED	PRODUCTION	\$11,000.00

NF710: Field FORMAT=YYJUL

A new date field formatting option, FORMAT=YYJUL, lets you print a Julian date in the format YYYY/DDD. The 7-digit format displays the four-digit year and the number of days counting from January 1. For example, January 3, 2001 in Julian format is 2001003.

FORMAT=JUL is still supported; however, only the last two digits of the year display (YY/DDD).

Example Displaying a Date in YYJUL Format

The example displays expiration dates in YYJUL format. It also illustrates the definition of a new date field that converts the Julian date and displays it in YYMD format.

```
FILENAME=EXAMPLE, SUFFIX=FOC
SEGNAME=ROOT, SEGTYPE=S1
FIELDNAME=DRIVER_ID, ALIAS=, FORMAT=A9, $
FIELDNAME=EXP_DATE, ALIAS=, FORMAT=YYJUL, $
FIELDNAME=CLASS, ALIAS=, FORMAT=A2, $

DEFINE FILE EXAMPLE
  NEWDATE/YYMD=EXP_DATE;
  END
TABLE FILE EXAMPLE
  PRINT EXP_DATE NEWDATE CLASS
  BY DRIVER_ID
END
```

PAGE 1

DRIVER_ID	EXP_DATE	NEWDATE	CLASS
123254365	2000/139	2000/05/18	A4
254069503	1999/068	1999/03/09	R4
675678904	2003/253	2003/09/10	W9

NF711: Altering Your System Date for Testing Purposes

TESTDATE is a new SET command that allows you to temporarily alter your FOCUS system date for a given application program. This allows you to determine what impact the year 2000 will have on your application programs.

Note: Only use TESTDATE for testing purposes with a test database.

The syntax for TESTDATE is:

```
SET TESTDATE = {yyyymmdd|TODAY}
```

where

yyyymmdd

Is an 8-digit date in the format YYYYMMDD.

TODAY

Is today's date. TODAY is the default.

The value of TESTDATE affects all reserved variables that retrieve the current date from the system. Setting TESTDATE also affects anywhere in FOCUS that a date is used (such as CREATE, MODIFY, MAINTAIN) but does not affect the date referenced directly from the system.

TESTDATE can either be equal to TODAY or a date in the format YYYYMMDD. If anything else is entered the following message is displayed:

```
TESTDATE MUST BE YYYYMMDD OR TODAY
```

NF713: MSO Log Changes

The MSO logs now display dates with four-digit years.

Sample MSO Log

Following is a portion of a sample MSOPRINT log. Note that the dates display with four-digit years:

```
06/10/1998 12.12.13 [MSD13137] PROCESSING EDACFG CONFIGURATION FILE
06/10/1998 12.12.13 [MSD13137] PROCESSING FOO.MSD CONFIGURATION FILE
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 1 -----
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 2 * Sample MSD configuration file.
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 3 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 4 * This file is allocated to ddname FOO.MSD
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 5 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 6 * The records and options which may be sp
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 7 * documented in the MSD Installation and
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 8 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 9 * Not all options which may be specified
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 10 * file; the options and values listed are
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 11 * be tailored to meet your installation's
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 12 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 13 * Please note that some of the options wh
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 14 * sample require AFP authorization and/or
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 15 * steps; please check the manual for futr
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 16 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 17 -----
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 18
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 19 -----
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 20 * Global processing and options:
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 21 -----
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 22 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 23 * Enable use of an external security packa
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 24 * EXTSEC = NO
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 25 EXTSEC = YES
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 26 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 27 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 28 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 29 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 30 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 31 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 32 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 33 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 34 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 35 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 36 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 37 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 38 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 39 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 40 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 41 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 42 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 43 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 44 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 45 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 46 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 47 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 48 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 49 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 50 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 51 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 52 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 53 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 54 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 55 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 56 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 57 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 58 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 59 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 60 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 61 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 62 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 63 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 64 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 65 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 66 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 67 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 68 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 69 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 70 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 71 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 72 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 73 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 74 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 75 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 76 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 77 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 78 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 79 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 80 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 81 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 82 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 83 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 84 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 85 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 86 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 87 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 88 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 89 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 90 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 91 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 92 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 93 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 94 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 95 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 96 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 97 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 98 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 99 *
06/10/1998 12.12.14 [MSD13136] CONFIG LINE ... 100 *
```

NF714: LE Support

You can now control use of IBM's Dynamic Language Environment for IBI-supplied and user-written subroutines by setting an option in FOCPARM, in a FOCUS application, or in your FOCUS session.

Syntax How to Control the LE Environment Setting

The syntax for specifying the type of LE support you need is

```
SET IBMLE = {OFF|ON|ALL}
```

where:

OFF

Does not initialize the LE environment. OFF is the default value and is the recommended setting for applications using only IBI-supplied subroutines.

ON

Establishes the LE pre-initialization environment with the IBM default configuration. This configuration initializes the LE environment for subroutines coded in COBOL, PL/I, C, and ASMH if the routines are linked with the LE environment. If the application calls a module not supported under LE, it runs without LE. For a list of languages supported under LE, see [LE Language Support for User-Written Subroutines](#).

ON is the recommended setting for applications that call user-written subroutines linked with the LE environment and not coded in FORTRAN. ON is also recommended for applications that call a combination of these subroutines and IBI-supplied subroutines. Running IBI-supplied subroutines with this setting requires LE version 1.5 or above.

ALL

Should be used only for user-created FORTRAN subroutines that need the LE environment. **Note:** This is not supported by IBM. The ALL setting adds FORTRAN to the list of languages supported for LE pre-initialization. The FORTRAN run-time libraries must be installed under LE. However, FORTRAN modules do not run under LE. ALL is the recommended setting for applications that call user-written subroutines written in FORTRAN if the FORTRAN run-time libraries were installed under LE. This setting requires LE version 1.5 and above.

Reference LE Language Support for User-Written Subroutines

FOCUS supports LE access to user-written subroutines coded in the following LE-supported languages:

- C/MVS™.
- COBOL for MVS & VM (Release 2).
- COBOL/370™ (Release 1).
- VS FORTRAN (Version 2).
- PL/I for MVS & VM.
- ASMH (with macro support).

Recommended IBMLE Settings

LE pre-initialization may be beneficial for application performance when using 3GL user-written subroutines. However, users should be aware that IBM may modify the LE environment at any time; its use is the responsibility of the user based on system analysis and resource requirements. IBM provides extensive documentation regarding LE at the following URL: www1.s390.ibm.com/os390.

Use the guidelines in the following table to determine the appropriate setting for each FOCUS application. If you need to change the setting in a particular application or in a FOCUS session, issue the SET IBMLE command prior to executing the first subroutine call. Results can be unpredictable if you change the setting between FOCEXECs or subroutine calls.

The application calls ...	Setting
At least one FORTRAN module, and the FORTRAN runtime libraries were installed under LE.	ALL
Modules linked with the LE environment and coded in languages other than FORTRAN.	ON
Only IBI-supplied subroutines and modules not linked with LE.	OFF

Determining Proper IBMLE Settings

The following table describes the results of various IBMLE settings on supported LE subroutines:

Type of Subroutine	Required Setting	Effects of Changing the IBMLE Setting
IBI-supplied	any setting	None, works with any IBMLE setting.
FORTRAN/LE	ALL	ALL is the only supported setting for FORTRAN/LE. Requires LE release 1.5 or above.

COBOL/LE	ON or ALL	<p>For performance reasons, link-edit the subroutine as reusable (<code>reus</code>).</p> <p>If the subroutine must be executed with <code>IBMLE=OFF</code>, apply the COBOL run-time option <code>rtereus</code> to the routine.</p> <p>Without this option, the subroutine returns invalid data and generates the following messages:</p> <pre>IGZ0044S There was an attempt to call the COBOL main program [xxxxxxx] that was not in initial state. The traceback information could not be determined.</pre> <p>The job does not abend but produces a CEEDUMP prefaced by:</p> <pre>CEE3DMP[release]: Condition processing resulted in the unhandled condition.</pre>
PL/I - LE	ON or ALL	<p>If executed with <code>IBMLE=OFF</code>, results in an 0C1 abend.</p>

Note:

- Non-LE Assembler subroutines require source code changes in order to be LE compliant.
- Mixed-mode applications calling both LE and non-LE subroutines in the same FOCEXEC or FOCUS session are not supported and may produce unpredictable results.

7.0.8 New Features

Year 2000

Project 2000 - Phase III

NF605: Date Handling for the Year 2000 in FOCUS

NF620: Year 2000 Subroutines

Performance Enhancements

NF564: Pooled Tables

NF593: IUCV CMS SU

NF617: Automatic Allocation of FOCUS Files

Raised Limits

NF642: Increased DEFINE Limitation

Reporting Enhancements

NF579: Assigning Screening Conditions to a File for Reporting Purposes

NF586: Expanding Byte Precision for COUNT and LIST

NF623: Increasing the Number of Verbs in a Report Request

General Enhancements

[NF607: TABLA Enhancements](#)

[NF609: Sink Validation of Userids in CMS](#)

[NF630: Querying Which PTFs Have Been Applied for a Specific Release](#)

[NF631: Extended Plists](#)

[NF640: Dynamic Language Environment \(LE\) Support](#)

The Multi-Session Option

[NF566: MSO/CICS Cooperative Processing](#)

Web Interface for FOCUS

[NF619: -HTMLFORM SAVE](#)

[NF626: JAVA Graph Wizard](#)

[NF594: JAVA Report Assist](#)

[NF628: Automatic Activation of Web Interface for Web Browser Users](#)

[NF645: WEBHOME](#)

Relational Interfaces

[NF568: DB2 Interface IF-THEN-ELSE Optimization](#)

[NF571: DB2 Interface SET ISOLATION Command](#)

Model 204 Interface

[NF572: Invisible Ordered Character and Ordered Numeric Data Type Key Support](#)

IMS Interface

[NF550: EDA/MSO Console Display for IMS PSB](#)

System 2000 Interface

[NF574: System 2000 Interface Trace Facility](#)

Teradata Interface

[NF583: Teradata Outer Join Optimization](#)

National Language Support

[NF647: Extended Support for Scandinavian External Sort](#)

NF550: EDA/MSO Console Display for IMS PSB

The EDA/MSO Console for the DU (Display Users) screen includes a new column heading named **PSB**. This column displays the name of the IMS PSB scheduled for each TCB that is accessing IMS. IMS PSB names are up to 8 characters in length.

Usage

When an EDA/MSO user subtask is accessing IMS, a PSB is used. By selecting DU from the Console and scrolling the display to the third screen to the right, a column named PSB will display the IMS PSB scheduled by a TCB. This is especially useful when a user subtask is not responding due to a runaway query. If the operator decides to, he can cancel that particular subtask. In the case where an IMS error condition is encountered, the PSB column remains populated; however, an asterisk (*) is placed in the first position as an indicator of an error condition. This can be useful for debugging purposes and/or notification of other users attempting to use that particular PSB.

Example Sample Console DU Screen

Below is an example of the CONSOLE DU screen display:

```
<PMSLCCM JOB04558----- .CONSOLE DISPLAY USERS PANEL. -----
Line:001(002) Col:153
COMMAND ==>

C Logon_ID Runcount MVSDD USER-DD OPN DDs QueryID1 QueryID2 PSB
*REGION*
USER01 ALLPSB
```

In the above example, userid USER01 has scheduled a PSB named ALLPSB. If an error was encountered, the PSB display for the user would have read:
***LLPSB.**

Reference Special Considerations

The PSB name will be populated at the time that the PSB is scheduled by the user subtask, and is removed after the DBCTL thread connection is successfully de-allocated. If an IMS error is encountered, the PSB column will remain populated until another PSB is scheduled. The PSB column is cleared whenever an END THREAD command is processed, or an OPEN thread is attempted. This feature requires no action to be activated.

Error Messages

None.

NF564: Pooled Tables

Pooled Tables is a FOCUS performance feature for reporting applications that enables many reports or extract files to be created from a single pass of a database. Requests from any database, file, or JOINed structure that FOCUS reads can be pooled, reducing all of the normal reporting costs including database I/O, CPU and elapsed time. Performance gains with Pooled Tables can be dramatic; there is no penalty for its use -- even with applications that do not take advantage of it.

Pooled Tables is simple to use: by just adding a few lines to your application, Pooled Tables does the work of identifying reports that can share database I/O and running them concurrently.

Pooled Tables is ideal for large applications with many reports and batch reporting applications. Additionally, reports where data retrieval costs are significant compared to formatting costs benefit greatly from Pooled Tables. There are several additional efficiencies that users can employ to maximize performance gains achievable through Pooled Tables.

This document describes how to use Pooled Tables. Refer to the *Pooled Tables White Paper* (DN1100978.0498) for additional information describing the internal logic.

This document covers the following topics:

- Overview - a general overview of Pooled Tables.
- Memory Needs - covers memory requirements for Pooled Tables.
- Report Size Estimates - describes record and line estimates for Pooled Tables.
- FOCPOOLT - describes a temporary work file Pooled Tables sometimes creates.
- Reporting Statistics - describes changes to the ? STAT output.

- Sort Selection - describes criteria for sort selection.
- Managing Memory - describes how you can control memory usage.
- Common Selection Criteria - describes efficiencies developers can employ.
- Reporting from Non-Relational databases - provides general information for using Pooled Tables with Non-Relational databases.
- Reporting from Relational databases - provides general information for using Pooled Tables with Relational databases.
- Trace Facility - describes the Pooled Tables trace facility.
- Pooled Tables in Batch Mode - provides general information for using Pooled Tables in batch mode.
- Tuning applications - provides suggestions for using Pooled Tables more efficiently.
- Syntax - provides syntax reference.
- Pooled Tables Example - illustrates the use of Pooled Tables commands.
- Single TABLE Clusters - describes situations when pooling is not done.
- Subpool Boundary Conditions - describes situations that constitute boundaries.
- Pooled Tables Installation Instructions
- Usage Notes
- Frequently Asked Questions
- Error Processing

Overview

Pooled Tables should be used whenever two or more consecutive reports are executed against the same database. It is ideal for use with large batch or canned FOCUS reporting runs and data-extract applications. The feature is implemented with only a few simple SET commands. There is no need to change an application. A sample program that uses Pooled Tables is illustrated in the section [Pooled Tables Example](#).

A pool begins with the command `SET POOL=ON` and continues until `SET POOL=OFF` is encountered. Within a pool, FOCUS reads ahead in an application searching for consecutive TABLE requests that access the same file using the same access method. This read ahead feature extends across FOCEXECs and divides commands into retrieval and non-retrieval categories called subpools. A subpool is a collection of TABLE requests and related commands. Only report requests within a subpool can be combined.

Commands that alter data or the processing environment create subpool boundaries. For example, in the sequence

```
TABLE FILE CAR
... END
TABLE FILE CAR
... END
MAINTAIN FILE CAR
... END
TABLE FILE CAR
... END
```

a subpool boundary occurs at the MAINTAIN command. A list of subpool boundary commands appears in the section [Subpool Boundary Conditions](#).

Subpools are further divided into clusters. A cluster is a set of consecutive TABLE requests that share the same logical database and access method. For example, two TABLE requests against a VSAM file, one using sequential access and the other using indexed retrieval, are placed in separate clusters because of the different access methods. Reports that cannot be pooled because of syntax or environmental conditions are executed as single TABLE clusters. A list of these conditions appears in the section “*Single TABLE Clusters.*”

All TABLE requests in a single cluster are executed concurrently and share the same data retrieval and screening processes. Sorting and output formatting are not shared.

The figure below diagrams the breakdown of pools into sub-pools and clusters:


```
SET POOL=ON  
  
TABLE FILE EMPLOYEE  
END  
TABLE FILE EMPLOYEE      CLUSTER  
END  
TABLE FILE EMPLOYEE      SUBPOOL  
END  
  
TABLE FILE CAR  
END  
TABLE FILE CAR           CLUSTER  
END  
-RUN  
  
TABLE FILE CAR  
END  
CLUSTER                 SUBPOOL  
-RUN  
  
TABLE FILE EMPLOYEE  
END  
TABLE FILE EMPLOYEE      CLUSTER  
END  
  
SET POOL=OFF
```

POOL

Memory Needs

The number of reports that can be executed in a single cluster is limited only by the amount of memory the user allocates to Pooled Tables (POOLMEMORY). More reports can be run concurrently with Pooled Tables when larger amounts of memory are available. Memory requirements for each report depend on the number of records included in the report, the number of lines of output, and the width of the report. Pooled Tables calculates these memory requirements. In general, the memory needed for small summary reports can be estimated as `NUMBER OF LINES OF OUTPUT * REPORT WIDTH`. The memory needed for large summary reports and detail reports can be estimated as `NUMBER OF RECORDS SELECTED * REPORT WIDTH`. The Pooled Tables trace facility displays the actual amount of memory allocated for each report and the statistics used to calculate it.

When available memory is insufficient for simultaneously executing all of the requests in a cluster, Pooled Tables executes them in a series of steps, called iterations.

When multiple iterations are required, Pooled Tables produces as many reports as it can directly in memory during the first iteration. Concurrently, data for the remaining reports in the cluster are staged in a work file called FOCPOOLT. The rest of the reports are then produced from the data in FOCPOOLT in subsequent iterations. The source database is only accessed once at the beginning of the process.

Report Size Estimates

To calculate memory needs, Pooled Tables requires accurate estimates of the size of each report to deliver optimal performance. These estimates are used to select the appropriate sort and distribute memory resources equitably across the several reports in the pool. Input report size equals the number of records in the report following selection (ESTRECORDS); output report size is the number of output lines after aggregation is complete (ESTLINES). These estimates apply to the individual reports, not the size of the set of reports in the cluster.

ESTLINES and ESTRECORDS estimates can be gathered from:

- The statistical message: NUMBER OF RECORDS IN TABLE= LINES=
- The RECORDS and LINES information available on the ? STAT output.
- Previously gathered information from the &RECORDS and &LINES variables.
- When ACROSS is used, ESTLINES = number of lines X number of unique ACROSS columns.
- When IF TOTAL or WHERE TOTAL is used, ESTLINES is the number of lines before the TOTAL selection is made.

These estimates should be set individually for every request. Global settings can be issued, however. If ESTRECORDS is set for a group of requests, the estimate should be representative of the most common reports and need not exceed the size of the database.

FOCPOOLT

The temporary work file FOCPOOLT is created only when a cluster contains more reports than can be executed in available memory (POOLMEMORY). If a cluster can be produced directly from memory, the FOCPOOLT file is not created.

For example, a cluster has 30 reports, each of which requires 1 megabyte of memory. There are 10 megabytes of memory available (POOLMEMORY). Pooled Tables retrieves all of the data once and produces the first 10 reports from memory (this is the first iteration). The records for the remaining 20 reports are written to the work file FOCPOOLT. In the second iteration, Pooled Tables reads data for the next 10 reports from FOCPOOLT and produces them. The final 10 reports are produced in the third iteration.

It is more efficient to get data from FOCPOOLT than the database, because data in FOCPOOLT has already been screened and formatted. In addition, Pooled Tables determines accurate record counts (ESTRECORDS) for all reports in the second and subsequent iterations. Memory needs for these reports are more accurately calculated, further optimizing Pooled Tables performance.

The size of FOCPOOLT depends on the volume of data in the reports that are executed in the second and subsequent iterations. Data required only for reports in the first iteration are not stored in FOCPOOLT. Overlapping data required for more than one report is stored only once. The size of FOCPOOLT will never exceed the size of the logical database used for the cluster. Typical size requirements for FOCPOOLT are the same as those for the largest FOCSORT file for any report in the pool.

For MVS users, the default allocation for FOCPOOLT is 5 primary and 20 secondary cylinders. FOCUS uses a second volume if all extents on the first volume are used. It is recommended that the user pre-allocate FOCPOOLT with the necessary space attributes under MVS. DCB information will be determined by FOCUS.

In CMS, adequate temp disk space must be made available for this file.

Reporting statistics

? STAT has been enhanced to display pooling statistics for each pooled report. It can be used to identify pooling characteristics in an application and tune the application.

Below is an annotated sample of the output for ? STAT with only the information for Pooled Tables shown.

		STATISTICS OF LAST COMMAND			
	RECORDS	=	50000	.	
	LINES	=	50000	.	
	.			.	
	.			.	
	.			.	
1.	READS	=	250000	.	
	.			.	
	.			.	
	.			.	
	.			.	
2.	SUBPOOL	=	1		
3.	CLUSTER	=	2	ITERATION	= 1 8.
4.	#CLUSTER ITEMS	=	25	#ITER ITEMS	= 16 9.
5.	SEQ# IN CLUSTER	=	5	SEQ# IN ITER	= 5 10.
6.	ESTIMATED RECS	=	50000	ESTIMATED LINES	= 50000 11.
7.	REPORT WIDTH	=	148		

Where, for the report just run:

1. `READS` The total number of records retrieved for the cluster.
2. `SUBPOOL` The report is in the first subpool.
3. `CLUSTER` The report is in the second cluster.
4. `# CLUSTER ITEMS` There are 25 reports in the cluster.
5. `SEQ# IN CLUSTER` This is the fifth report in the cluster.
6. `ESTIMATED RECS` ESTRECORDS has been set to 50,000. This number should be compared with RECORDS at the top of this ? STAT. If there is a discrepancy, change the ESTRECORDS value for this report.
7. `REPORT WIDTH` The report width is 148 bytes.
8. `ITERATION` This report was produced in the first iteration.
9. `# ITER ITEMS` There are 16 reports produced in the first iteration. The next 9 reports are executed during subsequent iterations.
10. `SEQ# IN ITER` This is the fifth report in this iteration.
11. `ESTIMATED LINES` ESTLINES has been set to 50,000. This number should be compared with LINES at the top of this ? STAT. If there is a discrepancy, change the ESTLINES value for this report.

Sort Selection

Pooled Tables uses the report size estimates to choose the appropriate sort: an in-memory FOCUS sort or an external sort. The FOCUS sort is used for all reports whose memory needs are less than 1 megabyte. In general, an external sort is used for all other cases. The maximum number of concurrently executing sorts, and thus the maximum number of concurrently executing reports, is limited by the amount of memory available to Pooled Tables. The maximum number of external sorts that can be used by Pooled Tables is 26. This number can be decreased with the MAXEXTSRTS setting. The number of external sorts can also be limited by the amount of available memory below the 16 megabyte line in MVS. In VM, only one version of the external sort can be executed when the sort package is SyncSort. When practical, the FOCUS sort is substituted for the external sort when the number of external sorts is limited but memory is available.

Managing Memory

The maximum amount of memory used by Pooled Tables can be limited with the POOLMEMORY setting. In MVS, the number represents memory above the 16 megabyte line. In VM, the number represents total virtual memory. The default value for POOLMEMORY is 16,384 K (16 M). The minimum value is 1,024 K. A maximum bound can be placed on POOLMEMORY when Pooled Table is installed. In MVS, you can also control the total amount of memory available from the operating system above the 16 megabyte line by coding REGION=nM in your JCL job card , where n is greater than 16. POOLMEMORY can be set from the command line, during FOCUS initialization (in the PROFILE FOCEXEC), or within an application.

Memory is reserved by using the POOLRESERVE setting. This reserves a portion of available memory for system or FOCUS use that is not to be used by Pooled Tables. In MVS, the default is 100K. In VM the default is 1,024 K. The default value can be changed at installation time. POOLRESERVE can be set from the command line, during FOCUS initialization (in the PROFILE FOCEXEC), or within an application.

The purpose of POOLRESERVE is to reserve memory during Pooled Tables' parsing and decision making process for other modules. For example, first time access to SQL/DS requires loading of IBI interface code and IBM modules. The memory needed for these is not used in the Pooled Tables case until the common read is executed. After these modules are loaded, POOLRESERVE can be reduced, possibly to zero. If the IBI interface and IBM load modules are stored in a saved segment, POOLRESERVE can be reduced prior to execution of Pooled Tables.

Suggested values for POOLRESERVE are :

```
Running an interface (not in saved segment): 1024 K
Running an interface (in saved segment):      256 K
Using SyncSort as the external sort:          512 K
Using any other sort:                          128 K
```

Common Selection Criteria

Common selection statements that appear in every report in the cluster are applied during Pooled Tables retrieval. The common test must refer to the same field and use an equality screening relational operator (`EQ` or `IS`). The selected values do not need to be the same in all reports. For example, if the first report has the test `WHERE FISCAL_YEAR EQ 1997` and the second request has the test `WHERE FISCAL_YEAR EQ 1998`, the test `WHERE FISCAL_YEAR EQ 1997 OR 1998` is evaluated during Pooled Tables retrieval. Common selection tests are included to reduce the size of the answer set returned for a pool.

Common selection criteria that do not use equality can be evaluated by Pooled Tables using another FOCUS feature: Filters. Filters allow you to specify simple or complex selection tests for all reports against the same file. Filters in effect for all reports in a cluster are also applied during Pooled Tables retrieval. The use of Filters allows you to reduce the size of the pooled answer set, even when there are no common equality selection tests.

Reporting from non-Relational Databases

Reports against non-relational databases, such as VSAM, IMS, IDMS, FOCUS, and sequential files, must meet several simple criteria in order to be pooled together into one cluster. First, all reports must access the same database, using the same Master File Description. Next, the reports in a cluster must share the same access method. For example, reports that use sequential access can be pooled together; reports that use indexed access can be pooled together. Finally, all reports in a cluster must share the same entry point. That is, the reporting view must be from the same segment and, in the case of indexed access, from the same field. Reports against sequential files always meet these criteria so they always pool. Reports against JOINed structures are pooled together based on the access method to the host file.

Reporting from Relational Databases

Reports against relational databases, such as DB2 and SQL/DS, can be pooled into the same cluster when they share several common attributes. Like non-relational files, all reports must access the same Master File Description from the same entry point. Reports that require SQL aggregation (that is, the generated SQL statements contain the `GROUP BY` phrase) are not pooled. This assures that the set presented to each report in the pool is accurate.

Further, all requests against a multi-table relational view must reference the same tables to be pooled into the same cluster. Consider, for example, a view that contains table 'A' and table 'B'. All reports that reference only fields from table 'A' can be pooled together; all reports that reference only fields from table 'B' can be pooled together, and all reports that reference fields from both tables 'A' and 'B' can be pooled together. However, none of the reports in each of these three preceding sets can be pooled with reports from another set. This limitation is imposed to assure that the same optimization logic is used by the RDBMS retrieval engine for each report in the set.

Pooling requirements for relational databases are less stringent when optimization is turned off (`SQL SET OPTIMIZATION OFF`). In this case, FOCUS manages the retrieval and aggregation. Therefore, pooling conditions are the same with optimization off as for non-relational databases. Restrictions regarding common accessed tables and SQL aggregation do not apply. The benefits of pooling reports with optimization off versus allowing the RDBMS to optimize retrieval vary from case to case. For example, a request that requires an area sweep and returns a large answer set, even with optimization, would be a good candidate to pool with other requests by turning optimization off.

When using the interface trace facility for a relational database, the generated SQL for each request is echoed. The SQL is generated during the Pooled Tables parsing phase but is not submitted to the RDBMS. Instead, Pooled Tables constructs an internal request to retrieve all of the data required for the cluster. The SQL SELECT statements generated for the cluster are echoed in the trace. These are the statements that are passed to the RDBMS.

The SQL SELECT statements generated by Pooled Tables are the ones optimized by the RDBMS. Therefore, the best optimization occurs when all requests in a cluster contain the same equality screening conditions or Filters (see [Common Selection Criteria](#)). In these cases, the screening tests are included in the SQL and passed to the RDBMS for optimization. Without common selections or Filters, it is possible that efficiencies gained from RDBMS optimization may be lost when pooling individual requests. For example, consider two requests: the first request returns a small answer set based on a selection against a key field named KEY1. The second request returns a small answer set based on a selection against a different key field named KEY2. The independent screening conditions are not included in the SQL generated by Pooled Tables, resulting in an area sweep and large answer set for the cluster. If the two tests are included as an OR condition in a Filter, the screening tests will be passed to the RDBMS. A much smaller answer set will be returned to Pooled Tables.

Pooled Tables in Batch Mode

Pooled Tables can automatically pool all batch requests. For batch jobs to become pools, issue the SET command POOLBATCH, from either users PROFILE or in FOC Parm. Wherever possible, pooling automatically occurs. In the context of Pooled Tables, 'batch' means any non-interactive FOCUS session. In MVS, this occurs in batch jobs, or when ddname SYSIN is allocated to a dataset. In VM, a non-interactive job occurs when ddname SYSIN is FILEDEFed to a file, FOCUS is invoked with the syntax `FOCUS IN fileid`, or the VM session is running disconnected.

Trace Facility

In general, the trace facility displays the reasons for segregation of a pool into subpools and clusters, warnings when allocating insufficient memory, and completion statistics for pooled reports. The purpose of the trace facility is to assist the application developer in determining how a pool was executed so that the information can be used in tuning the application.

The trace facility is started by issuing the command `SET TRACEON=POOLTABL`. By default, the trace output is routed to the ddname `PTTRACE` which is allocated to `SYSOUT` (MVS) or the terminal (VM). You can select a different ddname by issuing the command `SET TRACEON=POOLTABL//ddname`. To route the output to disk, allocate or `FILEDEF` ddname `PTTRACE` (or the optional ddname you selected) to a file with `LRECL 160` and disposition `MOD`. You may also allocate the ddname to the terminal. Stop the trace by issuing the command `SET TRACEOFF=POOLTABL`.

The following messages are displayed when a subpool boundary is encountered:

```
Subpool boundary--prior output required as input
Subpool boundary--FOCUS/SET command
Subpool boundary--DEFINE ADD
Subpool boundary--new MASTER name
Subpool boundary--new DEFINE clears pre-pool DEFINE
This command will run now, outside of pooling:
A DEFINE ADD will run now, outside of pooling.
```

The following messages are displayed when a cluster boundary is encountered:

```
Cluster boundary--new master name
Cluster boundary--single-table cluster
Cluster boundary--new alternate view
Cluster boundary--new pool flag
Cluster boundary--new pool condition
Cluster boundary--mid-stream DEFINE
Cluster boundary--new entry segment
Cluster boundary--too many verb objects
```

The following messages are displayed for reports that cannot be pooled (they are single-table clusters):

```
Single-table cluster--REDEFINED real field
Single-table cluster--User subroutine not known safe
Single-table cluster--self-referential DBA/filter
Single-table cluster--INCLUDES/EXCLUDES selection
Single-table cluster--too many test literals
Single-table cluster--complex test on index
Single-table cluster--$ORTPARM allocated
Single-table cluster--REDEFINED constant real field
Single-table cluster--RANKED BY
Single-table cluster--COUNT DISTINCT
Single-table cluster--RECAP
Single-table cluster--COUNT is a verb object
Single-table cluster--indexed view via AUTOINDEX
Single-table cluster--EMR
Single-table cluster--ON TABLE SET
Single-table cluster--TEXT field
Single-table cluster--PREVIEW mode
Single-table cluster--ALL = ON/PASS
Single-table cluster--per message above
Single-table cluster--indexed view for FOCUS database
Single-table cluster--non-poolable interface request
Single-table cluster--too many verb objects
```

The following messages appear in the trace during the creation and execution of clusters and iterations:

```
Building cluster x...
Cluster contains n table(s)
Cluster n dedicated to command x
Clusters built; subpool contains x cluster(s).
***** Stack before 1st cluster: *****
***** Stack before nth cluster: *****
***** Begin union table *****
**** Stack before nth iteration: ****
```

During the parsing phase of the Pooled Table process, the following statistics are displayed for each report. They are used to determine if a report is poolable and under what conditions. All reports that have the same pooling criteria can be pooled with each other.

```
Entry Segment   : x
Relational Flag : y
Pool Flag       : z
Condition Length: n
Condition       : c
```

After a pooled report is executed, the output from ? STAT is included in the trace. The entries for TRACKIO and MINIO are included in the output but their values are not populated. In addition, the following statistics are included:

```
TRAVERSAL MTHD =          x          ENTRY SEGMENT =          i
FOCUS SORT MEM =          y1         EXTSORT MEMORY =          y2
ALGORITHM USED =          z
```

The following messages are displayed to indicate limitations imposed on Pooled Tables to execute reports under the most favorable conditions, based on parameters provided by the user (POOLMEMORY, POOLRESERVE, ESTRECORDS, and ESTLINES) or the available memory. These messages will not inhibit the execution of Pooled Tables. To correct these situations, replace the values for ESTRECORDS and ESTLINES with accurate values or make more memory available for Pooled Tables.

```
# concurrent external sorts reduced from x to y by below-16M shortage
Minimum sort memory forces iterations
Warning--POOLMEMORY desired = x but only y is available
Warning: actual line count (x) exceeds lines estimate (y) in heavy
aggregation case
Warning: records estimate (x) off by more than 10%-actual record count=y
Warning: lines estimate (x) off by more than 10%-actual line count = y
```

Tuning Applications

Pooled Tables will always pool any application when POOL is set ON. Pooled Tables works best when accurate estimates for ESTRECORDS, ESTLINES, and POOLMEMORY are given for each request. These numbers can be determined by reviewing the statistics from previous runs. If these estimates are not provided, FOCUS uses the defaults: ESTRECORDS=100000, ESTLINES=0, and POOLMEMORY=16,384K. When ESTLINES is 0, Pooled Tables uses the current value of ESTRECORDS for ESTLINES. While these defaults are adequate for large extract reports, they may provide minimal benefit if they are grossly inaccurate.

To optimize pooling capacity, provide ample memory to Pooled Tables. Increase POOLMEMORY to an adequate size. Provide a sufficient region size (MVS) or virtual memory (VM). Reduce POOLRESERVE once interface and other modules are loaded. Furnish accurate estimates for ESTRECORDS and ESTLINES.

To optimize pooling capability, remove all unnecessary subpool boundary commands such as extraneous -RUNs. Consolidate the necessary boundary commands such as the DYNAMs, SETs, etc. Organize the requests for optimal cluster usage by putting all requests for the same database with the same entry point and retrieval method together. This will increase the opportunity for pooling more requests in one cluster.

To optimize retrieval and reduce the size of the answer set returned by Pooled Tables, use Filters to screen data. For example, if all reports in a cluster use `WHERE DELETE_FLAG NE 'Y'`, create a filter with this test. Alternately, change the test to read `WHERE DELETE_FLAG EQ 'N'` so that the common selection statement is used in the Pooled Tables common read.

Syntax How to Use Pooled Tables

To activate Pooled Tables, issue the following command

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

where:

`OFF`

Ends Pooled Tables and executes any queued requests.

`ON`

Begins Pooled Tables.

Issue the following command in a report request to supply an estimate for the number of input records for that report:

```
ON TABLE SET ESTRECORDS m
```

where:

m

Is the estimate of the number of records being retrieved for a report.

You can assign a global value for each report in a pool with the following command

```
SET ESTRECORDS=m
```

The default value is 100,000.

Issue the following command in a report request to supply an estimate for the number of output lines for that report:

```
ON TABLE SET ESTLINES n
```

where:

n

Is the user's estimate of the number lines of output for a report.

You can assign a global value for each report in a pool with the following command

```
SET ESTLINES=n
```

The default value is 0. If no value is given, Pooled Tables assumes there is no aggregation and the number of lines is the same as the number of records.

To set a limit on the amount of memory that FOCUS can use for pooling a cluster for a user, issue the following command

```
SET POOLMEMORY=n
```

where:

n

Is the upper limit on the number of kilobytes of memory that FOCUS may use during any cluster for this user. In MVS, the number represents memory above the 16 megabyte line. In VM, the number represents total virtual memory.

The default value is 16,384 K (16 M). The minimum value is 1,024 K.

To reserve memory for other modules, issue the following command

```
SET POOLRESERVE =n
```

where:

n

Is the amount of memory in kilobytes to reserve for other modules and restrict Pooled Tables from using.

In VM, the default is 1,024K. In MVS, the default is 100K.

To set a limit on the number of concurrent external sorts that can run, issue the following command

```
SET MAXEXTSRSTS=n
```

where:

n

Is the number of concurrent external sorts that can run.

Is a number from 1 to 26. The default is 26. In VM, only one version of SyncSort can run concurrently. If you use SyncSort in VM, the value of MAXEXTSRSTS is assumed to be 1.

To control whether Pooled Tables is used automatically for batch processing, issue the following command

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

where:

OFF

Does not enable automatic use of Pooled Tables for batch processing. This is the default.

POOLBATCH can be included in the FOC Parm ERRORS, FOCUS PROFILE, a FOCEXEC, or issued in the SYSIN input stream.

SET POOLBATCH=ON has the effect of automatically setting POOL=ON for batch execution. SET POOLBATCH=OFF will not reverse this setting. To disable pooling when POOLBATCH=ON, issue the command SET POOL=OFF.

ON

Enables automatic use of Pooled Tables for batch processing.

To identify an external sort utility to use for Pooled Tables, issue the following command

```
SET SORTLIB = sorttype
```

where:

sorttype

Can be one of the following

<code>SYNCSORT</code>	Identifies the external sort utility as SYNCSORT.
<code>DFSORT</code>	Identifies the external sort utility as DFSORT.
<code>VMSORT</code>	Identifies the external sort utility as VMSORT.
<code>MVSMGSS</code>	Identifies the external sort utility as SYNCSORT and its messages are displayed (MVS only).
<code>MVSMGDF</code>	Identifies the external sort utility as DFSORT and its messages are displayed (MVS only).

To direct the trace output, issue the following command

```
SET TRACEON=POOLTABL // {PTTRACE | ddname}
```

where:

`PTTRACE`

Is the default *ddname* where the trace output is directed.

ddname

Is an optional *ddname* where the trace output can be directed.

To turn off the trace facility, issue the following command

```
SET TRACEOFF=POOLTABL
```

where:

```
POOLTABL
```

Ends the Pooled Tables Trace facility.

Pooled Tables Example

The following example illustrates the ease in which Pooled Tables can be implemented. In it, a small amount of memory is made available for Pooled Tables (4,000K), pooling is turned on, and report size estimates are provided for each report. The reports will be queued until pooling is turned off. At that time, data will be retrieved only once for all of the reports in the pool. The reports will be executed concurrently and the output printed one after the other.

```
SET POOLMEMORY = 4000
SET POOL = ON
TABLE FILE EMPLOYEE
PRINT LN FN BY DPT IF HIRE_DATE GE 860101
ON TABLE SET ESTLINES 1000 AND ESTRECORDS 1000
END
TABLE FILE EMPLOYEE
SUM CURR_SAL BY CURR_JOBCODE IF CURR_JOBCODE EQ 'A$*'
ON TABLE SET ESTLINES 5 AND ESTRECORDS 400
END
TABLE FILE EMPLOYEE
SUM GROSS BY PAY_DATE
IF PAY_DATE FROM 960101 TO 961231
ON TABLE SET ESTLINES 52 AND ESTRECORDS 1200
END
SET POOL = OFF
```

Single TABLE Clusters

There are several instances when reports will not be pooled because of syntactical or environmental conditions. The reports will be executed as single TABLE clusters. Reports that fall into this category include:

- TABLEF requests.
- MATCH requests.
- Extended Matrix Reports (EMR).
- Reports using SET ALL=ON or PASS.
- Reports against FOCUS databases using an explicit indexed view or an implicit indexed view via AUTOINDEX.
- Reports against relational databases where aggregation is passed to the DBMS.
- Reports which use MORE, ON field RECAP, COUNT DISTINCT, DST., INCLUDES, EXCLUDES, or COUNT as a verb object.
- Reports that use a redefined database field.
- Reports issued from the FOCUS command line.
- Reports that use a self-referential Filter or DBA value restriction.
- Reports that have more than 256 values in an equality IF or WHERE test.
- Reports executed when \$ORTPARM is allocated.
- Reports that use a user written subroutine except those found in Table 1. In general, subroutines that require initialization and are then reused are not poolable. Random number generator subroutines are an example of these.

ARGLEN	ATODBL	AYM	AYMD	BAR
BITSON	BITVAL	BYTVAL	CHGDAT	CHKFMT
CHKPCK	CTRAN	CTRFLD	DADMY	DADYM
DAMDY	DAMYD	DAYDM	DAYMD	DMOD
DOWK	DOWKL	DTDYD	DTDYM	DTMDY
DTMYD	DTYDM	DTYMD	EXP	FEXERR
FINDMEM	FMOD	FTOA	GETPDS	GETTOK
GETUSER	GREGDT	HEXBYT	HHMSS	IMOD
ITONUM	ITOPACK	ITDZ	JULDAT	LCWORD
LJUST	LOCASE	OVLAY	PARAG	PCKOUT
POSIT	RJUST	SOUNDEX	SUBSTR	TODAY
UFMT	UPCASE	YM		

Table 1. Poolable User Written Subroutines

Subpool Boundary Conditions

A subpool is a collection of TABLE or GRAPH requests and their related commands. Subpool boundaries are imposed by non-retrieval commands. Only reports within a subpool can be pooled together to share the same I/O. Commands that cause subpool boundaries can change the data or retrieval method for the database. Therefore, reports on either side of a subpool boundary cannot be pooled together reliably. When a subpool boundary command is encountered, pooling is temporarily stopped and all queued requests are executed.

A subpool boundary is created when:

- A FOCEXEC completes execution and control is returned to the command line.
- A -RUN or -EXIT command is issued in a FOCEXEC.
- A `DEFINE FILE filename ADD` command is issued.
- Any non-TABLE or GRAPH command is issued that could change the data (MAINTAIN, MODIFY, SQL), change the source of the data (DYNAM, USE), change the retrieval method (JOIN, PASS, FILTER), or change the operating environment (TSO, MVS, CMS), Table 2 lists the retrieval commands that are part of a subpool. Table 3 lists the commands that cause subpool boundaries.
- Any SET or ON TABLE SET command that can alter retrieval or the Pooled Tables environment. Table 4 lists the SET commands that cause subpool boundaries. SET commands that appear in ? SET ALL and not on this list will not cause a subpool boundary. This list is accurate for FOCUS release 7.0.8 and is subject to change in subsequent releases.

?	?F	?FF	CHECK	DEFINE
GRAPH	HELP	HOLD	OFFLINE	ONLINE
PCHOLD	REPLOTT	RETYPE	SAVB	SAVE
TABLE	TABLEF			

Table 2. Commands Included in a Subpool

ACE	ANALYSE	CALC	CMS	COMBINE
COMPILE	CREATE	DECRYPT	DYNAM	ENCRYPT
EX	EXEC	FILETALK	FILTER	FIN
FINISH	FIXPACK	FS	FSCAN	GRAPHTALK
JOIN	LET	LOAD	MAINTAIN	MATCH
MODIFY	MODIFYTALK	MPAINT	MVS	PASS
PLOTTALK	REBUILD	RECALC	REMOTE	RESTRICT
RUN	SCAN	SET	SQL	TABLETALK
TED	TSO	UNLOAD	USE	WINDOW
XFER				

Table 3. Commands That Cause Subpool Boundaries

ADABAS	AGRRATIO	ALL.	AUTOINDEX
AUTOPATH	AUTOSTRATEGY	AUTOTABLEF	BINS
BLKCALC	BYPANEL 2	BYSCROLL	CACHE
CALC	CALCMEMORY	CALCROWS	CALCWAIT
CARTESIAN	CDN	COLUMNSCROLL2	COMMIT
COMPUTE	CONSULTOPTN	CURRENCY	DATETIME
DEFCENT	ESTLINES 1	ESTRECORDS 1	EXTSORT
FIELDNAME	FILENAME	FIXRETRIEVE	FOCSTACK

FOC144 1	HIPERFOCUS	HTMLMODE	ICUFORM
IMPLIEDLOAD	IMS	LABELPROMPT	LANGUAGE
LE370	LOADLIMIT	LOOKGRAPH	MAXLRECL
MAXPOOLMEM	MINIO	MODE XXXXXX	MPRINT
PASS	POOL	POOLBATCH	POOLFEATURE
POOLMEMORY	POOLRESERVE	PREFIX	PREVIEW
PRINTPLUS 2	QUALCHAR	RECORDLIMIT 1	SAVEMATRIX 2
SHIFT	SM	SQLENGINE	SQLTCARTES
SQLTOPTTF	STYLEMODE	SUSI	SUTABSIZE
TCPIPINT	TEMP DISK	TERMINAL	TEXTFIELD
TRACKIO	TRMSD	TRMSW	TRMTYP
USER	WINPFKEY	XRETRIEVAL	YRTHRESH
3DGRAPH			

Table 4. SETs That Cause Subpool Boundaries

- 1 - Subpool boundary with SET only
- 2 - Subpool boundary with ON TABLE SET only

Pooled Tables Installation Instructions

This section describes installation instructions for all systems, IMS, MVS, and VM/CMS.

Procedure Installation Instructions for All Systems

Pooled Tables is enabled for your release of FOCUS by including the command `SET POOLFEATURE = ON` in FOCPARM. To disable Pooled Tables, include the command

```
SET POOLFEATURE = OFF
```

in the FOCPARM file. If there is no `SET POOLFEATURE` in FOCPARM, FOCUS assumes Pooled Tables is disabled.

The maximum amount of memory above 16 megabytes that can be requested by a user with the `SET POOLMEMORY` command can be restricted by including the `SET MAXPOOLMEM = n` command in FOCPARM.

To make `POOL = ON` the default for all batch jobs, include the command `SET POOLBATCH = ON`. This must follow the `SET POOLFEATURE = ON` command in FOCPARM.

Each of the commands is also included in the member FOCPARM in ERRORS.DATA (MVS) or the file FOCPARM ERRORS (CMS).

Procedure Installation Instructions for MVS

Include the `POOLFEATURE`, `POOLBATCH`, and `MAXPOOLMEM` commands in the member FOCPARM in ERRORS.DATA as outlined above. Refer to the MVS Installation Guide for FOCUS Release 7.0 (DN1000994.1097) or New Feature Memo 607, *TABLA Enhancements*, to change the default allocation for the file FOCPOOLT. If you use DFSort, refer to [Usage Notes](#) for information about a required IBM PTF.

Procedure Installation Instructions for VM/CMS

Include the POOLFEATURE, POOLBATCH, and MAXPOOLMEM commands in the file FOCPARM ERRORS as outlined above. Change the value of POOLRESERVE in FOCPARM ERRORS if appropriate for your installation. See [Managing Memory](#) for recommended values.

Commands for the FOCPARM file

To configure Pooled Tables, include the following commands in the FOCPARM file.

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

where:

OFF

Disables Pooled Tables for this FOCUS site.

ON

Enables Pooled Tables for this FOCUS site.

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

where:

OFF

Does not enable automatic use of Pooled Tables for batch processing. This is the default.

POOLBATCH can be included in the FOCPARM ERRORS, FOCUS PROFILE, a FOCEXEC, or issued in the SYSIN input stream.

`SET POOLBATCH=ON` has the effect of automatically setting `POOL=ON` for batch execution. `SET POOLBATCH=OFF` will not reverse this setting. To disable pooling when `POOLBATCH=ON`, issue the command `SET POOL=OFF`.

ON

Enables automatic use of Pooled Tables for batch processing.

SET MAXPOOLMEM = *n*

where:

n

Sets upper limit in Kilobytes of memory above 16 megabytes available for users to set in the SET POOLMEMORY command. The default is 32,768 K (32 M). Minimum is 1,024K.

Reference Usage Notes

- With pooling, there may be differences in the order of output records in unsorted reports (PRINT with no BYs).
- In MVS batch jobs that have set `MSG=ON`, the TABLE request appears twice in the output.
- ? SET ALL has been enhanced to display the values of Pooled Tables settings. No Pooled Tables settings have been added to ? SET.
- If it is needed, adequate temporary disk space must be made available for FOCPOOLT under CMS.
- If it is needed, it is recommended that the user pre-allocate FOCPOOLT with the necessary space attributes under MVS. DCB information will be determined by FOCUS.
- KX following the attention interrupt is not supported during a pooled request.
- Pooled Tables memory in MVS is generally restricted using the POOLMEMORY setting. Pooled Tables memory in VM is generally restricted by using the POOLRESERVE setting.
- An implied SET POOL=OFF is issued and all queued requests are executed when an explicit or implicit FIN command is encountered and POOL is still ON.

- When SyncSort is the external sort package, the ddname \$ORTPARM must not be allocated. If \$ORTPARM is allocated, pooling is disabled for **all** requests, not only those that require the external sort. A warning message is issued when pooling is attempted and \$ORTPARM is allocated.
- DFSort Release 13.0 has a limitation where only 10 sorts can be run concurrently in MVS. If you exceed this limit, DFSort will display the message:

```
ICE149A DFSORT IS NOT LICENSED FOR USE ON THIS SYSTEM.RETURN CODE 12,  
REASON CODE 4.
```

This will cause FOCUS to abend. Issue the command SET MAXEXTSRTS = 10 to temporarily avoid this symptom. This problem has been fixed by IBM with APAR OW29152. Order IBM PTF UW41671 if you are running SMS Release 1.3. Order IBM PTF UW41672 if you are running SMS Release 1.4.

Frequently Asked Questions

- *"How can I control how much memory Pooled Tables uses? I am afraid that users will abuse memory resources."*

The command SET POOLMEMORY=n, where n is in kilobytes, sets the upper bound of memory that FOCUS will use for pooling on a per user basis. The default is 16 Megabytes and the minimum is 1 Megabyte. A maximum limit per user can be established during Pooled Tables installation by including the command SET MAXPOOLMEM=n in the FOCPARM file. A user will not be able to request more memory than this limit.

- *"I already create a HOLD file from my database and then report from that. Why do I need Pooled Tables?"*

With Pooled Tables, it is possible to receive even better results than this technique, without any pre-planning on the developers part! Rather than create the HOLD file, Pooled Tables will read the data only once and pass it to each of the reports in the pool. There is no I/O to write the HOLD file and, more importantly, no I/O to read the HOLD file once for each of the reports in the pool. If you already use this technique, there is no immediate need to change your application to benefit from Pooled Tables. All of the reports from the HOLD file can be pooled. This will alleviate all but the I/O to create the HOLD file and one set of I/O to read the file once for all of the reports in the pool.

- *"I have reports in two separate FOCEXECs. Will they be pooled?"*

Pooling will occur across FOCEXECs, as long as there are no intermediate commands between them that would create a subpool boundary. The most common subpool boundary command that would be encountered in this situation is the use of -RUN or -EXIT in the first FOCEXEC. Although this is good practice in the non-Pooled Tables case, it will cause reports that could benefit from Pooled Tables to be executed separately, as they are without Pooled Tables.

- *"What happens if I provide incorrect estimates for the number of records and lines in a pooled report?"*

Your reports will still execute and you will still receive the benefit of reduced database I/O and the CPU associated with it. However, the report sorting and formatting costs may increase. With incorrect estimates, Pooled Tables may select the wrong sort or allocate too little or too much memory for the report within the pool. If too much memory is allocated, fewer reports can be executed concurrently. If too little memory is allocated, fewer records can be sorted in memory, causing additional sort work I/O. If a FOCUS sort is selected, you lose the benefit an external sort when it is more appropriate. Note, however, that Pooled Tables will have an accurate count of the selected records (ESTRECORDS) for reports in the second and subsequent iterations.

Reference Error Processing

- Any error detected during parsing causes pooled tables to flush the remaining commands through `SET POOL = OFF`.
- Any error detected during the retrieval phase will cause Pooled Tables to abort the FOCUS session. This occurrence will display both the error message that caused the problem and error message FOC1897. The following is an example of when this may occur:

```
(FOC1070) VALUE FOR JOIN 'FROM' FIELD OUT OF SEQUENCE. RETRIEVAL ENDED  
(FOC1897) FATAL ERROR DURING POOLED TABLES RETRIEVAL. FOCUS  
TERMINATING.
```

- Any error detected during the output phase for a given TABLE terminates processing for that TABLE. Output processing continues for subsequent TABLE requests.
- Line numbers and FOCEXEC name may be missing in the message:
ERROR AT OR NEAR LINE n IN PROCEDURE a

Reference Warning/Error Messages

FOCUS warning messages (normally generated without Pooled Tables) may appear twice when running with Pooled Tables active.

Below is a list of messages generated by Pooled Tables. These appear only when FOCUS is about to be terminated.

(FOC1897) **FATAL ERROR DURING POOLED TABLES RETRIEVAL. FOCUS
TERMINATING.**

An error was encountered in a report during Pooled Tables retrieval. FOCUS cannot recover from this error and will return to the host environment.

(FOC1899) **LOAD FAILED FOR EXTERNAL SORT**

In preparation for an external sort under Pooled Tables, FOCUS tried to LOAD the external-sort module. The LOAD failed. (This message is produced only under VM.)

(FOC1900) **NOT ENOUGH MEMORY FOR EXTERNAL SORT**

In preparation for an external sort, available memory was queried. Not enough memory is available for an external sort. Increase memory and execute the request again.

NF566: MSO/CICS Cooperative Processing

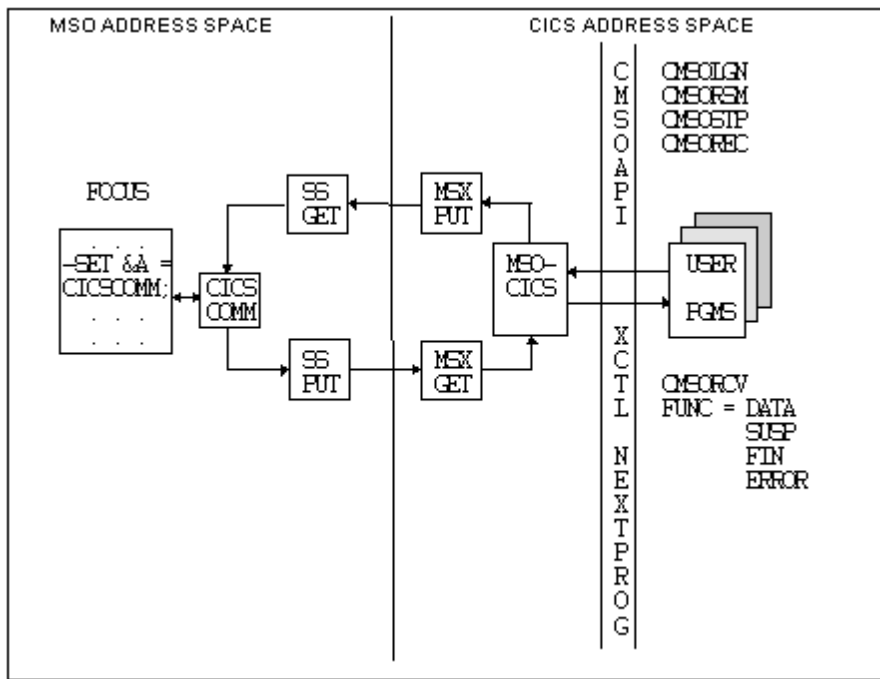
CICS transactions and MSO FOCEXECs may now communicate directly with each other in a synchronous mode. A CICS pseudo conversational transaction may start an MSO session on behalf of a CICS user. Once a cooperative processing session is started, data can be handed back and forth between FOCEXECs and CICS transactions in packets containing up to 256 bytes. Additionally, a suspend function is available. When this is invoked in MSO, the CICS transaction is given control. This allows a “hot” MSO to be available to the CICS user. The CICS user may go in and out of MSO without terminating their MSO session.

These new facilities allow CICS transactions to get and retrieve data from the MSO region and the reverse is also true. The FOCEXEC interaction is implemented in MSO via a FUSELIB routine, CICSCOMM.

The CICS transaction functions are implemented by linking the installation’s transaction module with the IBI supplied module, CMSOAPI. CMSOAPI provides five function calls that supply the MSO/CICS communications.

Also, a reconnect facility is available for cooperative processing sessions and for standard MSO connections from CICS. This allows the reconnection of an MSO session when the logical connection from a CICS terminal is lost.

The figure below illustrates the relationship between a FOCEXEC running in the MSO address space and user programs running in the CICS address space when a cooperative session has been established.



MSO FOCEXEC Cooperative Processing Service

The CICS_{COM} FUSELIB routine supplies the cooperative processing facility to FOCEXECs executing in the MSO region. It may be called from wherever a FOCUS user-written subroutine is supported. This routine is used to communicate and synchronize activity with the CICS portion of the dialog. When invoked, the FOCEXEC is placed into a wait until action is taken on the CICS side of the conversation. Data specified by the outlen/outbuf parameters is passed to CICS. When this subroutine is called it causes a CICS transaction specified in the CMSLOGN to be started in the CICS region. When CICS_{COM} completes, the inlen/inbuf parameters contain data that was passed back from CICS.

CICS_{COM} supports the following syntax:

```
CICSCOMM(timeout, outlen, outbuf, inlen, inbuf);
```

where:

timeout

Is the number of seconds to wait for response before timing out FOCUS. If this timeout duration is reached before the CICS portion of the session responds, the MSO FOCUS session is terminated. This action is represented as a FIN function code to the CMSRCV call.

outlen

Is length (0-256) of the outbound message to CICS.

outbuf

Is the field containing the outbound message to CICS.

inlen

Is the length (0-256) of the inbound message buffer.

This value is the maximum amount that can be returned by CICS. It is presented to the CICS portion of the conversation in the *outlen* parameter of the CMSORCV call.

inbuf

Is the field to contain the inbound message from CICS.

MSO/CICS Cooperative Processing Services

The CICS portion of the cooperative processing functions is provided in the IBI supplied module CMSOAPI. This module contains the functional code that supports the individual calls available to CICS transactions. When this module is link edited with a CICS transaction module, five functions become available to the module.

CMSOLGN	Logs a CICS user as an MSO FOCUS user and establishes a session between MSO FOCUS and CICS
CMSRCV	Interrogates MSO/CICS to identify the session that requires servicing, receiving data if sent.
CMSREC	Reconnects to a session that is in an indeterminant state.
CMSOSTP	Stops or terminates the session. This cancels the users MSO FOCUS task.
CMSORSM	Sends data back to the FOCUS portion of a session or restarts a suspended MSO FOCUS session.

Syntax and descriptions of the functions follow:

Syntax **How to Use the CMSOLGN Function**

This function starts an MSO session for the user by logging on to MSO. It defines the details of the MSO/CICS conversation processing that will take place.

```
CALL CMSOLGN(EIB, COMMAREA, MSONAME, USERWD, NEXTPROG, LOGONBUF)
```

where:

EIB

is the CICS Exec Interface control Block

COMMAREA

is the CICS Communications Area

MSONAME

4 byte name of MSO CICS transaction. Each MSO transaction name corresponds to a single MSO region that may be connected to.

USERWD

4 byte user word to associate with an MSO session

It is an arbitrary 4 byte value that is returned by CMSOLGN. It represents a unique identifier for the conversation that this service just established. It will be returned when the CMSRCV is issued to identify the specific conversation that needs to be serviced

NEXTPROG

8 byte name of next program to call

It identifies the CICS user program that will be called when the CMSCOMM FUSELIB is invoked on the MSO side of the conversation. NEXTPROG is mutually exclusive with the LOGTRAN entry in the LOGONBUF control block. Only one of these parameters should be specified. They function identically. Cooperative processing support, that is, the ability to be called back at all, is enabled by specifying a NEXTPROG in CMSOLGN. Fill NEXTPROG with blanks or nulls if no callbacks are desired.

LOGONBUF

Is a required control block. It should be completely initialized to blanks (x'40') before individual fields are set. Some of the fields support the MSO Load Balancing feature. The purpose of these fields are fully described in the new feature documentation for that feature.

An assembler copy file, CMSOAPIA, is supplied in MSO.DATA. This file maps the LOGONBUF. The fields and their meanings are listed in the table below. All fields are alphabetic.

Field Name	length	contents
LOGAPPL	8 bytes	A load balancing parameter that limits MSO region selection to those service groups that specify the same application name.
LOGSERV	8 bytes	Specifies the particular MSO Service Group that this user should be started in. It applies to load balancing as well as a single region MSO.
LOGTRAN	4 bytes	This defines the CICS transaction that will be invoked when the CICSCOMM FUSELIB routine is called in the MSO region. It is mutually exclusive with NEXTPROG in the invocation parameters. Only one should be specified.
LOGBREAK	4 bytes	Break key (PF/PAnn) Identifies the key that will unconditionally terminate the active MSO session.
LOGSUSP	4 bytes	Suspend key (PF/PAnn) Identifies the key that will cause the active MSO session to suspend operation. When this key is pressed, the LOGTRAN or NEXTPROG CICS transaction (whichever was specified) will be invoked in the CICS region.

Field Name	length	contents
LOGELVL	4 bytes	Error level (<i>ALL, ERR, NONE</i>) Controls what messages are displayed to the user when the MSO session ends. ALL - All messages are displayed ERR - Only error messages are displayed NONE - No messages are displayed
LOGGROUP	8 bytes	Logon group name The load balancing group name. It controls the MSO load balancing group that the user will be started in.
LOGVALID	1 byte	Security check flag (<i>N/Y</i>) Y must be specified for the reconnect service (CMSOREC call) to work. In addition, UNIQUE=LOGONID must be specified in the MSO configuration file.
LOGINITM	1 byte	Suppress initialization message flag (<i>N/Y</i>) Y suppresses the MSO initialization message.
LOGFLAG1	1 byte	reserved flag
LOGFLAG2	1 byte	reserved flag
LOGRESV	20 bytes	reserved

Field Name	length	contents
LOGACCT	40 bytes	account Specifies the MSO account field. This field will be recorded in the MSO SMF records when that feature is active.
LOGUPRM	256 bytes	Logon parameter The contents of this field is available to FOCEXECs via the MSOINFO subroutine. It is generally used to provide control information to the MSO profile exec so that specific FOCUS applications may be invoked in the MSO region.

Syntax How to Use the CMSORCV Function

This function is used to interrogate the MSO/CICS control program. The returned parameters identify the particular conversation that had a status change, the current status of the conversation, and any data that may have been received from MSO. It is usually the first MSO/CICS Cooperative Processing service used in the transaction that is triggered by the CICS COMM subroutine (NEXTPROG or LOGTRAN).

The **FUNCTION** field is set based on either an event in the MSO region or if an event in the CICS region caused the status. The **FUNCTION** codes that are returned are described in CMSORCV Function Codes.

```
CALL CMSORCV(EIB, COMMAREA, FUNCTION, CONNID, USERWD, INLEN, OUTLEN,  
BUFFER, ERRNUM)
```

where

EIB

Is the CICS Exec Interface control Block

COMMAREA

Is the CICS Communications Area

FUNCTION

4 byte callback function code See CMSORCV Function Codes

CONNID

4 byte connect id of MSO session. This value together with USERWD uniquely define each MSO/CICS session.

USERWD

4 byte user word to associate with an MSO session. This value together with CONNID uniquely define each MSO/CICS session.

INLEN

4 byte length of buffer inbound from FOCUS

OUTLEN

4 byte length of return buffer expected by FOCUS

When using the CMSORSM, the [LENGTH](#) parameter may not exceed the value of [OUTLEN](#). If it does, the data presented to the MSO FOCCEXEC is truncated to the value of [OUTLEN](#).

BUFFER

256 byte inbound data buffer

ERRNUM

4 byte FOCUS ending error number (function FIN) or 4 This field is a binary number and is mapped by the CMSOAPIA member of MSO.DATA. Possible returned values are described in section: CMSORCV Function Codes.

Syntax How to Use the CMSOREC Function

This function re-establishes a MSO/CICS session based upon the current user's CICS id. The connect id may be supplied if known. Otherwise, the function uses a connect id of 0 and the userid to identify the session. This condition may be caused by the user powering off their terminal while an MSO/CICS session is active and then logging on to CICS again.

Security flag - If the security flag in the logon buffer is set to yes, then the resuming or reconnecting userid is validated against the known userid. The reconnection is rejected if they do not match. If present userid determination is subject to the `MSCXUID` exit. The default for the security flag in the logon buffer is no.

```
CALL CMSOREC(EIB, COMMAREA, MSONAME, CONNID, NEXTPROG)
```

where:

`EIB`

is the CICS Exec Interface control Block

`COMMAREA`

Is the CICS Communications Area

`MSONAME`

4 byte name of MSO transaction

`CONNID`

4 byte connect id of MSO session

`NEXTPROG`

8 byte name of next program to call

Syntax How to Use the CMSOSTP Function

This function stops an MSO/CICS session immediately. MSO must have passed control to the CICSCOMM FUSELIB program before a stop can be issued. A `CMSOSTP` received while MSO is still in control is treated as a protocol error.

```
CALL CMSOSTP(EIB, COMMAREA, MSONAME, CONNID, NEXTPROG)
```

where:

`EIB`

Is the CICS Exec Interface control Block

`COMMAREA`

Is the CICS Communications Area

`MSONAME`

4 byte name of MSO transaction

`CONNID`

4 byte connect id of MSO session

`NEXTPROG`

8 byte name of next program to call. This field specifies a CICS transaction to start if the current invocation fails and cannot be associated with a known session. If a session is identified then the nextprog that was specified in the CMSOLGN service for the identified session will be called.

Syntax How to Use the CMSORSM Function

This function sends data to an MSO FOCUS session that previously issued the CICSCOMM FUSELIB routine or resumes an existing MSO/CICS session that was suspended by the user with the suspend key. Data may be sent in both cases but will be ignored if the session is in a suspended state. A `CMSORSM` received while MSO is still in control is a protocol error. If there is data, a buffer is allocated which is freed by MSOCICS.

CALL CMSORSM (EIB , COMMAREA , MSONAME , CONNID , NEXTPROG , LENGTH , BUFFER)

where:

EIB

Is the CICS Exec Interface control Block

COMMAREA

Is the CICS Communications Area

MSONAME

4 byte name of MSO transaction

CONNID

4 byte connect id of MSO session

NEXTPROG

8 byte name of next program to call

LENGTH

4 byte length of outbound buffer. Data sent to MSO FOCUS will be truncated to the original length specified by INLEN on the MSO FOCEXEC call to CICSCOMM.

BUFFER

Outbound data buffer (up to 256 bytes)

CMSORCV Function Codes

The possible function codes that may be returned on the CMSORCV call are listed below:

Value name	value	Meaning	Returned Parameters
IB_DATA	9	Data returned This is the result of CICSCOMM being invoked in MSO.	CONNID USERWD INLEN OUTLEN BUFFER
IB_SUSP	10	suspend key struck The users MSO session is dormant until CMSORSM is issued for it.	CONNID USERWD
IB_FIN	11	FOCUS session ended ERRNUM contains return code from the MSO FOCUS session.	CONNID USERWD ERRNUM
IB_ERROR	12	API protocol error ERRNUM contains the value for the error. See the following table for a list of errors and their meanings.	CONNID USERWD ERRNUM
IB_NOAPI	17	Program not called by MSO\CICS	None

The ERRNUM values that may be associated with a function code of IB_ERROR are:

Error	Value	Description
IBERR_STATE	1	MSO called in invalid state
IBERR_INVFUNC	2	MSO called w/invalid function
IBERR_SECURE	3	id verification failed.
IBERR_NORECON	4	RECON called but not supported
IBERR_NOTFOUND	5	user not found for connect id

Examples

The following sample members are supplied in MSO.DATA to aid in developing installation applications to use this feature:

CCDEMO	A FOCEXEC that implements a sample MSO menu to illustrate the function supplied by the CICSCOMM FUSELIB routine
CCDEMOAS	Assembler source of CICS API program
CCDEMOAJ	JCL to build CCDEMOA module
CCMAPS	map source for demo program
CCMAPSAJ	JCL to build mapset and MAP DSECT

Reconnection Capability

A CICS connection may be lost by powering off a terminal or closing an emulator session. This feature adds the capability of reconnecting to that session. Previously there was no way to do this and the user would not be allowed to re-login until the original session had timed out or been canceled by the MSO operator.

Now, a standard MSO session may be reconnected with, by specifying MSO RECON (where MSO is the CICS transaction that invokes the MSOCICS program).

An MSO/CICS cooperative processing session, one that was established via the CMSOLGN call, may be reconnected using the CMSREC call. CMSREC allows the use of conid or userid to be specified. If conid is specified then the userid of the MSO FOCUS session has to match the CICS userid only if the LOGVALID security flag was set to Y in the LOGONBUF. The reconnection is found using the userid associated with the CICS session. UNIQUE=LOGONID must have been specified in the MSO configuration file for this service to be able to reconnect.

Suspend key

If the suspend key is activated in the MSO FOCUS session, it is subject to the MSO configuration setting of IDLELIM. If the session is in the suspended state long enough to set off the IDLELIM limit, the session is terminated. This condition can be avoided by defining a separate service group that has IDLELIM set to a high value and placing the cooperative processing MSO sessions into these service groups by specifying the appropriate LOGSERV parameter.

Previous API

The CMSOLGN is an alternate method of starting an MSO/CICS session. The original method was documented with the Load Balancing new feature of FOCUS 7.0.5 & 7.0.6. Logon according to those specifications is still supported. More information is available in New Feature Bulletin NF554.

NF568: DB2 Interface IF-THEN-ELSE Optimization

The DB2 interface has been enabled to improve the performance of FOCUS TABLE requests that include IF/THEN/ELSE define statements. Where applicable, the defined statements will be passed to DB2 as expressions allowing DB2 to optimize its own execution and minimize the size of the answer set returned to FOCUS.

Usage

By issuing the new DB2 interface set command OPTIFTHENELSE, the interface will attempt to deliver as an expression to DB2 the construct of FOCUS IF/THEN/ELSE defines. The defined field must be an object of a selection test or an object of an aggregation request. The define definition may be specified in the table request or in the master file description.

Syntax How to Enable IF-THEN-ELSE Optimization

```
SQL {DB2} SET OPTIFTHENELSE {ON|OFF}
```

where

ON

Enables the feature

OFF

Disables the feature and is the default

Note: Omit the DB2 target RDBMS qualifier to issue the command if you previously issued the SET SQLENGINE command for DB2.

Example Using IF-THEN-ELSE Optimization Without Aggregation

```
SQL DB2 SET OPTIFTHENELSE ON
DEFINE FILE DB2TABLE
DEF1 = IF (NAME EQ ' ') AND (NAME EQ 'XYZ') AND (SSNO EQ ' ') THEN 1
ELSE 0;
END
TABLE FILE DB2TABLE
PRINT SSNO NAME
WHERE DEF1 EQ 1
END
>SELECT T1.SSNO,T1. NAME FROM Creator.table T1 WHERE
> (((((T1.TOTAL_NAME = ' ') AND (T1.NAME = 'XYZ')) AND
(T1.SSNO = ' ')))) FOR FETCH ONLY;
```

Example Using IF-THEN-ELSE Optimization With Aggregation

```
DEFINE FILE DB2TABLE
DEF2 = IF NAME EQ 'NAME1' THEN 1 ELSE IF NAME EQ 'NAME2' THEN 2
ELSE IF NAME EQ 'NAME3' THEN 3 ELSE 0 ;
END
TABLE FILE DB2TABLE
WRITE MAX.NAME IF DEF2 EQ 1
END
> SELECT MAX(T1. NAME) FROM creator.table T1 WHERE
(((T1. NAME = 'NAME1')) FOR FETCH ONLY;
TABLE FILE DB2TABLE
WRITE MAX.NAME IF DEF2 EQ 2
END
> SELECT MAX(T1. NAME) FROM creator.table T1 WHERE ((NOT
(T1. NAME = 'NAME1')) AND (T1. NAME = 'NAME2')) FOR FETCH ONLY;
```

Example Using IF-THEN-ELSE Optimization With Selection Criteria That Is Always False

```
DEFINE FILE DB2TABLE
DEF3=IF NAME EQ 'RITA' THEN 1 ELSE 0;
END

TABLE FILE DB2TABLE
PRINT NAME IF DEF3 EQ 2
END

>          SELECT T1.NAME FROM creator.table T1 WHERE (1 = 0) FOR FETCH ONLY;
```

Note: Please note that DEF3 EQ 2 will never be true, thus the interface passes the where test 1=0 to DB2, denoting a not true condition and returning zero records from DB2.

Reference Special Considerations

This new feature is enabled for SELECT statements only created as a result of FOCUS TABLE requests.

The following features are not supported:

- Decode defines
- Self-referencing or “recursive” defines
- STATIC SQL requests
- IF/WHERE DDname defines
- Partial Date selection

There is no guarantee that the SQL that is generated will improve performance for all requests. If it's found that this feature does not improve performance, OPTIFTHENELSE OFF will disable this feature to initiate previous behavior.

Error Messages

None.

NF571: DB2 Interface SET ISOLATION Command

Starting with FOCUS release 7.0.8, the interface has been enabled to take advantage of the DB2 version 4 (and higher) ability to pass an SQL statement isolation level. This allows the interface to modify the isolation level for a TABLE request with the new interface SET command. The SET ISOLATION command will override the isolation level that was established at interface installation bind time. The new setting removes the requirement of binding multiple DB2 plans, each bound with different isolation levels.

Usage

DB2 protects data being read by one user from changes (INSERT, UPDATE, or DELETE) made by other users; the isolation level setting governs the duration of the protection. The isolation level determines when shared locks on rows or data pages are released, so that rows or pages become available for updates by other users. DB2 version 4 and above allows this isolation setting to be changed at the SQL statement level. Besides the earlier DB2 supported isolation levels of CS and RR, DB2 version 4 and above has introduced the uncommitted read isolation level (UR) and in DB2 5.1, introduced the read stability isolation level (RS) and the isolation level of NC.

Please refer to the *DB2 Read/Write Interface Users Manual* for a more detailed definition of isolation level and to the appropriate DB2 manuals for a complete description of CS, RR, UR, RS and NC isolation levels and their impact on concurrency and database implications.

Syntax How to Set the DB2 Isolation Level

```
SQL {DB2} SET ISOLATION level
```

where:

level

Can be one of the following DB2 isolation levels:

RR

UR

RS

NC

Omit the DB2 target RDBMS qualifier to issue the command if you previously issued the SET SQLENGINE command for DB2.

To display the isolation level setting, issue the command SQL DB2 ?.

Issue the command SQL DB2 SET ISOLATION (without an isolation level) to revert back to the bound isolation level of the interface plan.

Example Setting the DB2 Isolation Level

This example shows the SQL passed to DB2 as a result of the SET ISOLATION command:

```
SQL SET ISOLATION UR
SELECT T1.field1 FROM "creator"."table1" T1 FOR FETCH ONLY WITH UR;
SQL SET ISOLATION CS
SELECT T1.field1 FROM "creator"."table1" T1 FOR FETCH ONLY WITH CS;
```

The following is a result of the SQL DB2 ? command:

```
(FOC1424) ISOLATION LEVEL FOR TABLE REQUEST IS : CS
```

To reset the isolation to the Interface default:

```
SQL SET ISOLATION (blank)
SELECT T1.field1 FROM "creator"."table1" T1 FOR FETCH ONLY;
```

The following is a result of the SQL DB2 ? command:

```
(FOC1424) ISOLATION LEVEL FOR TABLE REQUEST IS :
```

The blank isolation level denotes the default isolation level will be used.

Reference Special Considerations

This new feature is enabled for SELECT requests only created as a result of FOCUS TABLE requests.

This new setting cannot be used to override the required isolation level of RR for FOCUS MODIFY requests, COPY MANAGER or CACTUS/MAINTAIN applications.

RS isolation level is only available for DB2 version 5.1 (4.2).

The interface does not validate the isolation level values, so be sure that they are one of the acceptable isolation levels for the version of DB2 that is being accessed, otherwise an SQL code of -104 will occur signifying a SQL syntax error.

Error Messages

None.

NF572: Invisible Ordered Character and Ordered Numeric Data Type Key Support

This new feature will allow access and selection of Model 204 invisible ordered character and invisible ordered numeric fields. The interface supports these key designations with comparable abbreviations called *suffix operators*. These operators are described with the TYPE attribute in the Access File Description.

Usage

A TYPE=IOA in the Access File will designate a Model 204 Key of invisible ordered character. A TYPE=ION would denote an invisible ordered numeric field.

Any FOCUS or EDA selection request against a field with TYPE=IOA or ION will result in a Model 204 IFFIND command with the appropriate IFFIND specification.

Example Using TYPE=IOA and TYPE=ION to Produce an IFFIND Specification

For example, an equality test produces the following IFFIND specification:

Suffix Operator	Model 204 KEY Type	IFFIND Specification
IOA	Ordered Character, invisible	IS ALPHA
ION	Ordered Numeric, invisible	IS NUM

NF572: Invisible Ordered Character and Ordered Numeric Data Type Key

A range test using LT or GT produces the following IFFIND specification:

Suffix Operator	Model 204 KEY Type	IFFIND Specification
IOA	Ordered Character, invisible	IS ALPHA BEFORE/AFTER
ION	Ordered Numeric, invisible	IS NUM LT/GT

A range test using FROM-TO produces the following IFFIND specification:

Suffix Operator	Model 204 KEY Type	IFFIND Specification
IOA	Ordered Character, invisible	IS ALPHA BEFORE/AFTER
ION	Ordered Numeric, invisible	IS NUM LT/GT

Special Considerations

For a complete list and chart of all Model 204 KEY types and their corresponding IFFIND specifications, please refer to the *Model 204 Interface Users Manual*.

Error Messages

N/A

NF574: System 2000 Interface Trace Facility

Two new trace levels have been enabled. The traces can be used for informational or debugging purposes. You can access the SYSTEM 2000 database two ways with FOCUS or EDA: Selective or Sequential. FSTRACE will display all SYSTEM 2000 calls made by the Interface. If the selective strategy has been used, then FSTRACE1 can be used to display the SYSTEM 2000 LOCATE command with the associated parameters.

Usage

You can store the trace information in an MVS sequential data set or CMS file, to SYSOUT in a batch job, or you can display it online at the terminal.

Syntax How to Invoke the System 2000 Interface Trace Facility

For Online to the screen (do not use for EDA or MSO):

```
DYNAM ALLOC F(FSTRACE) DA(*)
```

To write the trace to a file, use the appropriate allocation:

```
MVS ALLOC F(FSTRACE) DA('userid.FSTRACE') SHR REUSE LRECL(80) RECFM(F)
TSO ALLOC F(FSTRACE) DA('userid.FSTRACE') SHR REUSE LRECL(80) RECFM(F)
DYNAM ALLOC FILE FSTRACE DATASET userid.FSTRACE SHR REUSE LRECL 80 RECFM
F
```

For a batch job, to write the trace information to SYSOUT:

```
//FSTRACE DD SYSOUT=*,DCB=(LRECL=*,BLKSIZE=80,RECFM=F)
```

For CMS:

```
CMS FILEDEF FSTRACE DISK FSTRACE DATA A (LRECL 80 RECFM F
```

Note: Depending on the trace desired, specify FSTRACE or FSTRACE1. You must specify the MOD parameter in order to produce a complete trace listing. The FSTRACE and FSTRACE1 data sets are opened and closed for each SYSTEM 2000 request. Without the MOD disposition parameter, requests that produce multiple database accesses store only the last statement in the dataset.

Reference Special Consideration

FSTRACE1 will contain less information than FSTRACE since it is restricted to SYSTEM 2000 LOCATE calls. It should contain enough information to determine interface to database communication and/or possible causes of performance degradation. When utilizing FSTRACE, it is recommended to use a RECORDLIMIT of 1 to begin with and increase this limit when needed since the FSTRACE file produced can be quite voluminous.

Error Messages

N/A

NF579: Assigning Screening Conditions to a File for Reporting Purposes

A filtering mechanism that assigns screening conditions to a file has been added to the functionality of TABLE. This enables you to declare a set of screening conditions, and assign it to a specific file, instead of constantly having to rewrite these screening conditions every time you need them.

The following example illustrates the use of filters. Both sides yield the same results.

Without filters

```
TABLE FILE CAR...  
WHERE...SEATS GT 5  
  
TABLE FILE CAR...  
WHERE SEATS GT 5...  
  
TABLE FILE CAR...  
WHERE SEATS GT 5
```

With Filters

```
FILTER FILE CAR...  
SET FILTER= WHERE SEATS GT 5...  
  
TABLE FILE CAR...  
TABLE FILE CAR...  
TABLE FILE CAR...  
TABLE FILE CAR...
```

Whenever a TABLE request is made against a file, all filters that have been activated for that file are in effect. A filter is a packet of definitions that resides at the file level, containing IF and/or WHERE statements. Once these conditions have been declared, you may deactivate and reactivate them as needed for your reporting purposes.

Using Filters

A filter is an IF or WHERE statement that is automatically added to every TABLE against that file as if the IF or WHERE were actually coded by the user. All IF/WHERE syntax that is valid in a TABLE is valid in a filter. A filter can be declared at any time before the TABLE request, and remains in effect after the TABLE request has been executed. There can be one or more filters declared for a file.

Just declaring a filter for a file does not make it active. A filter must be activated with a SET command to be in effect.

Filters allow you to:

- Declare a common set of screening conditions that apply to all extracts from a file.
- Declare a set of screening conditions and dynamically turn them on and off.
- Reduce repetitive ad hoc typing.
- Implement DBA capabilities that are not tied to the Master File Description.

Syntax How to Declare a Filter

A filter can be described by the following declaration:

```
FILTER FILE filename [CLEAR/ADD]
  [filter-defines;]
  NAME=filtername1 [,DESC=text]
  if-where-statements
  ...
  NAME=filternamen [,DESC=text]
  if-where-statements
END
```

where:

filename

Is the name of master to be used in subsequent TABLE commands

filter-defines

Are virtual fields declared for use in filters. For more information, see [Filter Defines](#).

NAME

Identifies the start of the declaration of a new filter

filtername

Is the name by which the filter is referenced in subsequent SET FILTER commands. Filtername may be up to 8 characters in length and must be unique for a particular filename.

CLEAR

Releases any previously declared filters.

ADD

Enables you to specify additional filters without releasing existing ones.

DESC

Describes the filter. Text must fit on one line for documentation purposes.

END

Terminates the filter

if-where-statements

Are screening conditions that can include all valid syntax. May not refer to defined fields declared via DEFINE FILE. May refer to Database fields, defines in the Master. May not refer to other filternames.

Reference Filter Defines

- Are exclusively local to (usable by) filters in the filter block.
- Are not referenceable by DEFINE FILE DEFINES or TABLE.
- Support any syntax valid for DEFINES in DEFINE FILE.
- Cannot reference DEFINES from DEFINE FILE but can reference DEFINES in the Master File Description.
- Do not count toward the 256 verb object limit of TABLE unlike DEFINES from DEFINE FILE when referenced explicitly or implicitly.
- Must all be declared before the first named filter.
- Must each end with a semi-colon
- Cannot be enclosed between DEFINE FILE/END commands.

Example Using Filters

The following example replaces the filter UK, with a new WHERE condition. It also adds to the CAR file's set of filter-defines, a definition for "MARK_UP". When the TABLE command is issued for CAR, and UK is activated, the condition WHERE MARK_UP is greater than 1000 is automatically added to the TABLE request.

Note: The field MARK_UP cannot be explicitly displayed or referenced in the TABLE..

```
FILTER FILE CAR ADD
MARK_UP/A16=RCOST-DCOST;
NAME=UK
WHERE MARKUP GT 1000
END
TABLE FILE CAR
PRINT
```

The following example declares three named filters for the CAR file; ASIA, UK, and LUXURY. The filter ASIA is given a textual description, for documentation purposes only. The CLEAR on the first line erases any named filters that had existed for CAR, as well any filter DEFINES for CAR, before it processes the new definitions.

```
FILTER FILE CAR CLEAR
NAME=ASIA,DESC=Asian cars only
IF COUNTRY EQ JAPAN
NAME=UK
IF COUNTRY EQ ENGLAND
NAME=LUXURY
IF RETAIL_COST GT 50000
END
```

Syntax **How to Activate/Deactivate Filters**

Filters can be activated and deactivated with the following SET command:

```
SET FILTER {*|xx[ yy zz]} IN file {ON|OFF}
```

where:

*

Denotes all declared filters (default)

ON

Activates the filter.

OFF

Deactivates the filter (default).

xx

Is the name of a filter as declared in the NAME = syntax of the FILTER FILE block

The following is an example of filter activation and deactivation:

```
SET FILTER = UK LUXURY IN CAR ON
...
TABLE FILE CAR
PRINT COUNTRY NAME MODEL RETAIL_COST
END
...
SET FILTER = LUXURY IN CAR OFF
TABLE FILE CAR
PRINT COUNTRY NAME MODEL RETAIL_COST
END
```

The first SET FILTER activates CAR's filters, UK and LUXURY, and applies the conditions their filters contain to any subsequent TABLE request of CAR. The second SET FILTER, deactivates the filter named LUXURY of CAR. Any subsequent TABLE request (unless LUXURY is activated again) of CAR will not apply the conditions in LUXURY but continues to apply UK.

Syntax Filter Query

In order to find out the status of any existing filters, use the following syntax:

```
? FILTER [{file|*}] [SET] [ALL]
```

where:

file

Is the name of a database master

*

Displays filters for all files that have filters declared

SET

Displays only active filters

ALL

Displays all information about the filter including its description and the exact IF/WHERE definition.

Example Querying Filters

The following is an example of querying filters:

```
? FILTER
NO FILTERS DEFINED
```

or

```
Set File      Filter name Description
-----
   CAR        ROB          Rob's selections
*  CAR        PETER         Peter's selections for CAR
*  EMPLOYEE   DAVE         Dave's tests
   EMPLOYEE   BRAD         Brad's tests
```

```
? FILTER CAR
NO FILTERS DEFINED FOR FILE NAMED CAR
```

or

```
Set File      Filter name Description
-----
   CAR        ROB          Rob's selections
*  CAR        PETER         Peter's selections for CAR
```

```
? FILTER * SET
Set File      Filter name Description
-----
*  CAR        PETER         Peter's selections for CAR
*  EMPLOYEE   DAVE         Dave's tests
```

Filters and JOINS

Filters against a file are suspended (but not erased) when that file is the object of a JOIN. Filters against the primary file of a JOIN may be re-declared or be entirely different, may reference only fields in the JOINed structure and are in effect until the JOIN is cleared. At that time the pre-JOIN filters are brought back. Filters against the secondary JOIN file(s) remain alive as originally declared.

NF579: Assigning Screening Conditions to a File for Reporting Purposes

```
_*****
-* JOIN AND FILTER INTERACTION
_*****

-* DECLARE A FILTER
FILTER FILE EMPLOYEE CLEAR
  NAME=XXX WHERE JOBCODE EQ 'A01'
END
SET FILTER = XXX IN EMPLOYEE ON
-* EMPLOYEE FILE SHOWS JOBCODE A01 ONLY
TABLE FILE EMPLOYEE PRINT EMP_ID JOBCODE
END
-* -----
-* NOW JOIN TO JOBFILE AND REDECLARE THE SAME FILTER TO A DIFFERENT VALUE
-* -----
JOIN JOBCODE IN EMPLOYEE TO JOBCODE IN JOBFILE
FILTER FILE EMPLOYEE
  NAME=XXX WHERE JOBCODE EQ 'A07'
END
-* (NOTE: NEW FILTER FOR JOIN STRUCTURE IS NOT ACTIVATED YET)
-* EMPLOYEE FILE SHOWS **ALL** JOBCODES (ORIGINAL FILTER TURNED OFF BY
JOIN)
TABLE FILE EMPLOYEE PRINT EMP_ID JOBCODE
END
-* -----
-* NOW TURN ON THE NEW FILTER THAT APPLIES TO THE JOIN STRUCTURE
-* -----
SET FILTER = XXX IN EMPLOYEE ON
-* SHOWS JOBCODE A07 (NOT A01) (NEW FILTER APPLIES TO JOIN ONLY)
TABLE FILE EMPLOYEE PRINT EMP_ID JOBCODE
END
-* NOW CLEAR THE JOIN TO RE-ESTABLISH THE ORIGINAL FILTER
JOIN CLEAR *
-* NOW SHOWS JOBCODE A01 ONLY, AS BEFORE (ORIGINAL FILTER REACTIVATED)
TABLE FILE EMPLOYEE PRINT EMP_ID JOBCODE
END
```

Reference Special Consideration

The maximum number of filters set 'ON' for a file is limited by the number of IF/WHERE statements in these filters, not to exceed the standard FOCUS limit of IF/WHERE statements in any single TABLE.

The SET FILTER command is limited to one line. To activate more filters than fit on one line, repeat the SET FILTER command. As long as you specify 'ON' the effect is additive, not one of replacement. For example,

```
SET FILTER A B C IN CAR ON
SET FILTER D E F IN CAR ON
SET FILTER G IN CAR OFF
```

activates A, B, C, D, E, F and deactivates G (assuming that it was set ON previously).

Reference Error Messages

```
(FOC 36237) SYNTAX ERROR SETTING FILTER
(FOC36241) FILTERS DON'T EXIST FOR FILE NAMED:
(FOC36242) FILTER DOESN'T EXIST
```

NF583: Teradata Outer Join Optimization

This feature improves the Teradata Relational Interface performance by enabling the interface to deliver better optimized SQL to the Teradata RDBMS, permitting the RDBMS to optimize its own join processing.

Usage

The interface now passes left outer joins to Teradata when you issue the FOCUS command

```
SET ALL=ON
```

(**Note:** SET ALL=ON previously disabled optimization, leaving FOCUS to handle the join).

Optimization of outer joins is available for the interface installed with Teradata TOS version 1 release 5.1 and higher or Teradata version 2 release 2 and higher. The Teradata version is specified by the REL= parameter of the Teradata Installation procedure GENFDBC.

Syntax How to Invoke Teradata Outer Join Optimization

```
SET ALL=ON
```

```
SQL SET OPTIMIZATION ON
```

Example Invoking Teradata Outer Join Optimization

This example shows the SQL passed to Teradata for a FOCUS dynamic join with SET ALL=ON, SQL SET OPTIMIZATION ON:

```
JOIN field1 IN file1 TO field2 IN file25
SELECT T1.field1,T2.field2 FROM "creator.table1" T1 LEFT OUTER JOIN
"creator.table2" T2 ON T2.field2 = T1.field1 FOR FETCH ONLY;
```

Special Considerations

- The SET ALL=ON command also controls processing of short paths. The interface default setting is OFF. For a complete description of short path processing, please refer to your Interface Users Guide.
- Please be aware that in passing requests to Teradata with SET ALL=ON, you may get correct, yet subtly different, sequences of report rows than you had with earlier releases of the Interface. These differences are due to differences between sorting algorithms used by FOCUS and by Teradata, and do not indicate upward compatibility problems.

Error Messages

N/A

NF586: Expanding Byte Precision for COUNT and LIST

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

Usage

Before FOCUS Release 7.0.8, if the number of records retrieved for a field exceeded 5 bytes, asterisks were displayed in the report. This indicates an overflow condition, meaning that the display must be increased. The new maximum value for COUNT and LIST is 999,999,999.

Syntax How to Set the Precision for COUNT and LIST

`SET COUNTWIDTH = ON/OFF`

where OFF is the default.

Example Setting Precision for COUNT and LIST

The following example shows the COUNT verb with behavior prior to FOCUS Release 7.0.8:

```
TABLE FILE filename
COUNT Fldxx
BY Fldyy
END
```

```
          FLDxx
Fldyy  COUNT
value  *****
```

The following example shows the COUNT verb with behavior as of FOCUS Release 7.0.8 with SET COUNTWIDTH = ON:

```
TABLE FILE filename
COUNT Fldxx
BY Fldyy
END

          FLDxx
Fldyy    COUNT
value   999999999
```

Special Considerations

This feature will affect the width of a report when the COUNTWIDTH is set to ON. Calculating the LRECL of a report will now require an additional 4 bytes for each COUNT and LIST column.

Error Messages

None.

NF593: IUCV CMS SU

Inter-user communications vehicle is now supported for CMS SU. This is a desirable protocol as the master processor is not enqueued serially per request. IUCV also offers performance gains when compared to the VMCF communications protocol.

IUCV is not supported for any release prior to R7.0.8. If any earlier release of FOCUS is used with an IUCV server the results are unpredictable. Please see the Simultaneous Usage Reference Manual, CMS Version (DN 1000015.0797)

NF594: JAVA Report Assist

Java Report Assist provides a user-friendly environment for creating ad hoc reports in HTML, WP, DF or Lotus formats. The Report Assistant supports automatic generation of complete record selection criteria, sort fields, headings and footings, subtotals, and calculations.

Complete documentation for the Web Interface product can be found in the Web Interface User's Manual and Installation Guide Release 7.0.8 (DN1001038.1097).

NF605: Date Handling for the Year 2000 in FOCUS

As part of our year 2000 compliance effort we have changed the default date format display in FOCUS. The new format is MMDDCCYY.

Usage

The two digit century, has been added to the year portion of the display. This applies to all areas within FOCUS that display a date. They include:

- The FOCUS Banner
- ? REL
- ? FILE
- ? FDT
- MODIFY FILE FN
- CREATE FILE FN
- FSCAN FILE FN
- REBUILD TIMESTAMP

A 4 digit year is written into the FOCUS file in the format of CCYY. Prior to Release 7.0.8, YY was written into page 1 of the FOCUS file.

Example Displaying Four-digit Years in FOCUS

Entering ? REL at the FOCUS prompt displays a screen similar to the following:

>

? REL

FOCUS 7.0.8

CREATED 11/20/1997

Date Literals Interpretation Table

This table illustrates the behavior of FOCUS date formats. The columns indicate the number of input digits for a date format. The rows indicate the usage or format of the field. The intersection of row and column describes the result of input and format.

	1	2	3	4
YYMD	*	*	CC00/0m/dd	CC00/mm/dd
MDYY	*	*	*	*
DMYY	*	*	*	*
YMD	*	*	CC00/0m/dd	CC00/mm/dd
MDY	*	*	*	*
DMY	*	*	*	*
YYM	CC00/0m	CC00/mm	CC0y/mm	CCyy/mm
MY Y	*	*	*	*
YM	CC00/0m	CC00/mm	CC0y/mm	CCyy/mm
MY	*	*	0m/CCyy	mm/CCyy
M	0m	mm	*	*
YYQ	CC00/q	CC0y/q	CCyy/q	0yyy/q
QYY	*	*	q/CCyy	*

NF605: Date Handling for the Year 2000 in FOCUS

YQ	CC00/q	CC0y/q	CCyy/q	0yyy/q
QY	*	*	q/CCyy	*
Q	q	*	*	*
JUL	CC00/00d	CC00/0dd	CC00/ddd	CC0y/ddd
YY	000y	00yy	0yyy	yyyy
Y	0y	yy	*	*
D	0d	dd	*	*
W	w	*	*	*

	5	6	7	8
YYMD	CC0y/mm/dd	CCyy/mm/dd	0yyy/mm/dd	yyyy/mm/dd
MDYY	0m/dd/CCyy	mm/dd/Ccyy	0m/dd/yyyy	mm/dd/yyyy
DMYY	0d/mm/CCyy	dd/mm/Ccyy	0d/mm/yyyy	dd/mm/yyyy
YMD	CC0y/mm/dd	CCyy/mm/dd	0yyy/mm/dd	yyyy/mm/dd
MDY	0m/dd/CCyy	mm/dd/Ccyy	0m/dd/yyyy	mm/dd/yyyy
DMY	0d/mm/CCyy	dd/mm/Ccyy	0d/mm/yyyy	dd/mm/yyyy
YYM	0yyy/mm	yyyy/mm	*	*
MYY	0m/yyyy	mm/yyyy	*	*

YM	0yyy/mm	yyyy/mm	*	*
MY	0m/yyyy	mm/yyyy	*	*
M	*	*	*	*
YYQ	yyyy/q	*	*	*
QYY	q/yyyy	*	*	*
YQ	yyyy/q	*	*	*
QY	q/yyyy	*	*	*
Q	*	*	*	*
JUL	CCyy/ddd	*	*	*
YY	*	*	*	*
Y	*	*	*	*
D	*	*	*	*
W	*	*	*	*

- CC stands for two century digits provided by DFC/YRT settings.
- * stands for error message FOC177 (invalid date constant).
- FOCUS reads date literals from right to left.

Special Considerations

N/A

Error Messages

(FOC177) `INVALID DATE CONSTANT:`

The constant in the calculation is not a valid date. Enter a valid date.

NF607: TABLA Enhancements

(Default Space Allocation Table for Work Files)

FOCUS output datasets not allocated by the user are allocated dynamically by FOCUS itself. The default space attributes associated with each dynamically allocated ddname are now set by editing the member IBITABLA in the PDS called FOCCTL.DATA.

Usage

In order to change the defaults for these FOCUS output datasets, the file IBITABLA must be copied to a dataset allocated to the DDname ERRORS. The file is a fixed columnar file. All changes must be made in the appropriate column. The columns are:

Column name	Starting column	length
DDname	01	8
Allocation units (<u>CYLS</u> ,TRKS)	10	4
Primary space	15	3
Secondary space	19	3
Number of Directory entries (PDS)	23	2
Sysout class (OFFLINE only)	26	1
Volume	28	6
Unit (SYSDA,DASD,HIPER,NOHIPER*,etc)	35	8
Unit Count (FOCPOOLT only)	44	2

- * Use NOHIPER as a unit name to exclude particular datasets from HiperFOCUS.

If the file IBITABLA is not available in a PDS allocated to the ddname ERRORS, the defaults are in affect for all FOCUS output datasets. The default for all FOCUS output datasets except FOCPOOLT is 5 CYLS with secondary extent size of 5. The default for FOCPOOLT is 5 CYLS with secondary extent size of 20.

Example Sample IBITABLA

This is a copy of IBITABLA as shipped.

```
* DDNAME*A.UN*SP1*SP2*DR*C*VOLUME* UNIT *UC* FIELD NAME
*0-----1----1----1---2--2-2-----3-----4-* STARTING
*1-----0-----5---9---3--6-8-----5-----4-* COLUMN
*---8-----4-- -3- -3- 2- 1 --6--- ---8----- 2-* LENGTH
HOLD CYLS 5 5 , /* 1 */
HOLDMAST TRKS 5 5 36 , /* 2 */
SAVE CYLS 5 5 , /* 3 */
REBUILD CYLS 5 5 , /* 4 */
FOCSML CYLS 5 5 , /* 5 */
FOCUS CYLS 5 5 , /* 6 */
FOCSTACK TRKS 5 5 , /* 7 */
FOCSORT CYLS 5 5 , /* 8 */
OFFLINE CYLS 5 5 A , /* 9 */
SESSION TRKS 5 5 , /*10 */
FOCCOMP TRKS 5 5 12 , /*11 */
HOLDACC TRKS 5 5 12 , /*12 */
FMU TRKS 5 5 12 , /*13 */
TRF TRKS 5 5 12 , /*14 */
FOCPOOLT CYLS 5 20 2, /*15 */
*
*
* The UC or unit count column may be specified for FOCPOOLT only.
***** Bottom of Data
*****
```

Special Consideration

This New Feature document supersedes Section 3.14 'Default Space Allocation Table for Work Files: TABLA' in the *FOCUS MVS/TSO Installation Guide* (DN1000994.0295, DN1000994.0896, or DN1000994.1097)._

Error Messages

If the old method is used and the installer attempts to link TABLA into module FOCUS, there is now a U593 abend at FOCUS initialization.

NF609: Sink Validation of Userids in CMS

A new file has been created in CMS to help verify who may connect to a specific CMS sinkid.

Usage

A filename called FOCSUACC, with FILETYPE = DATA and FILEMODE = A1, can be created on the A disk of the CMS sink id. The DCBs of the file are as follows: LRECL 80 RECFM F BLKSIZE 80. You may code up to eight-character userids in this file. They are coded one per line in the file. When HLIMAIN or IUCVMMAIN programs initialize, a search is performed for this file. If it's found, all of the userids stored in the file are read into a list in memory. This list is then used to verify who may connect to this CMS sink id with READ and/or WRITE access. If a client tries to connect to the sinkid and their userid doesn't exist in the FOCSUACC DATA file, an error message is displayed. This provides an equivalent of MVS SUSI for the CMS SU product.

Examples

Verification that this feature is working can be simply tested. For example, if FOCSUACC DATA A has two userids coded on two lines, and you start the sink, and attempt to connect with a third userid that is not coded in FOCSUACC DATA an error will display after the first request for data from the sink.

Special Considerations

ENCRYPT FILE is not supported for this feature.

Error Messages

(FOC517) SU. ACCESS DENIED BY EXTERNAL SECURITY SYSTEM:

NF617: Automatic Allocation of FOCUS Files

The automatic allocation of FOCUS files in MVS FOCUS was removed from FOCUS Release 7.0.1. This was a dramatic change from prior releases where TABLE, MATCH, MODIFY, etc. commands would search the catalog for 'prefix.master.FOCUS'. If the file is found, the allocation for the file would be made automatically. This change was made for performance reasons.

Usage

With the new SET command activated before running a TABLE, MODIFY, or MATCH, etc. request, FOCUS dynamically issues the equivalent of a DYNAM ALLOC or TSO ALLOC.

Syntax How to Activate or Deactivate Automatic Allocation of FOCUS Files

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

where OFF is the default.

Example Activating and Deactivating Automatic Allocation of FOCUS Files

The following shows a TABLE request and its result in any FOCUS Release between 7.0.1. and 7.0.7, without any previous allocation to the EMPLOYEE file:

```
TABLE FILE EMPLOYEE_  
PRINT EMP_ID  
END  
(FOC036) NO DATA FOUND FOR THE FOCUS FILE NAMED: EMPLOYEE
```

The same TABLE request in FOCUS Release 7.0.8, without any previous allocation to the EMPLOYEE file, but after setting FOCALLOC on, yields a report with all of the EMP_ID records in the EMPLOYEE file.

Reference Special Considerations

? SET ALL shows this feature in its list. ? SET does not show this feature in its list. This is an MVS only feature.

Error Messages

None.

NF619: -HTMLFORM SAVE

FOCUS 7.0.8 introduces a new -HTMLFORM SAVE feature for the WEB Interface. With this feature, you can save html content generated by the -HTMLFORM command to a file, rather than to the screen. All FOCUS amper variables (&var) and escape sequences (e.g., '!IBI.AMP.varname') will be fully resolved in the external file.

Usage

This feature may be useful for those wishing to use FOCUS to generate HTML content in batch mode, rather than interactively.

Syntax Using -HTMLFORM SAVE

```
-HTMLFORM BEGIN SAVE AS filename
```

or

```
-HTMLFORM htmlfile SAVE AS filename
```

where:

filename

Is a 1 to 8 character filename of the file that will receive the html output from -HTMLFORM.

htmlfile

Is a 1 to 8 character filename of a source file containing HTML that you wish to save as output of -HTMLFORM.

In CMS FOCUS, this creates an html file named filename with a filetype of HTML.

In MVS FOCUS (TSO or MSO), this creates an html extract file in one of three ways:

- If ddname 'filename' is allocated, to either a sequential file or a member of a partitioned dataset, the html is written to that file.
- If ddname 'filename' is not allocated, but ddname 'HTML' is allocated to a partitioned dataset, the file is written to the HTML PDS as member
- 'filename.'
- If neither 'filename' nor 'HTML' is allocated, the file is written to a temporary sequential file using ddname 'filename'.

In each case, this HTML file can then be copied to a Web server for display to end users' browsers.

Special Consideration

Complete documentation for the Web Interface product can be found in the Web Interface User's Manual and Installation Guide Release 7.0.8 (DN1001038.1097).

Error Messages

FOC36235 - AS KEYWORD NOT SPECIFIED FOLLOWING SAVE

NF620: Year 2000 Subroutines

Enhancements have been made to subroutines that handle dates. All subroutines that perform date calculations or perform date format conversions support dates including and after the year 2000. This change was made to coincide with our Year 2000 project.

The subroutines that were re-written for Year 2000 are:

AYMD	Adds and subtracts days from a date.
AYM	Adds and subtracts months from a date.
YM	Finds the number of months between two dates.
CHGDAT	Rearranges the year, month and day portions of dates, and connects dates between long and short formats.
JULDT	Converts dates in year-month-day format into Julian format.
GREGDT	Converts Julian dates to year-month-day dates.
DAxxx	Converts dates into number of days elapsed.
DTxxx	Converts the number of days elapsed into date usage.

Usage

The new versions of these date subroutines are used by default.

Syntax How to Choose a Version of a Subroutine

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

where:

ON

is the default.

If you require the older version which does not have Year 2000 capabilities, deactivate this feature by setting DATEFNS = OFF.

Example Using Subroutines With Year 2000 Capabilities

The following FOCEXEC shows a straight conversion on the input to the subroutine.

```
SET DEFCENT=19,YRTHRESH=50
  DYNAM ALLOC FILE DATE DS PMSPAK.DATE.FOCUS SHR REU
  TABLE FILE DATE
  ON TABLE SUBHEAD
  " THIS EXAMPLE ILLUSTRATES THE USE OF THE THIRD PARAMETER IN THE      "
  " SUBROUTINE CALL.  THE INPUT FIELD IS AN I6YMD FORMAT.  IT CONTAINS  "
  " 2 DIGIT YEARS.  SETTING A PIVOT YEAR OF 1950 RESULTS IN CONVERSION  "
  " TO A FULL 4 DIGIT YEAR VIA THE FORMAT OF 'I8'.                      "
  PRINT D2_I6YMD AND COMPUTE
  X/I8YYMD=AYMD(D2_I6YMD,1,'I8');
END
```

Output:

PAGE 1

THIS EXAMPLE ILLUSTRATES THE USE OF THE THIRD PARAMETER IN THE SUBROUTINE CALL. THE INPUT FIELD IS AN I6YMD FORMAT. IT CONTAINS 2 DIGIT YEARS. SETTING A PIVOT YEAR OF 1950 RESULTS IN CONVERSION TO A FULL 4 DIGIT YEAR VIA THE FORMAT OF 'I8'.

```
D2_I6YMD          X
-----          -
97/09/16  1997/09/17
00/02/29  2000/03/01
01/02/28  2001/03/01
00/02/28  2000/02/29
```

The following example demonstrates giving the subroutine a 4 digit year as input.

```
DYNAM ALLOC FILE DATE DS PMSPAK.DATE.FOCUS SHR REU
TABLE FILE DATE
ON TABLE SUBHEAD
" START WITH A COMPUTED FIELD CALLED A.  IT HAS A PIVOT YEAR OF 1950."
" CONVERTING IT FROM I6YMD TO YYMD SETS THE CENTURY DIGITS TO '20'  "
" FOR THOSE YEARS WHICH ARE LT 50.  CREATE OLD DATE FIELD B.      "
" PASS B INTO THE AYMD SUBROUTINE ADDING 1 DAY TO THE DATE.      "
PRINT D2_I6YMD AND COMPUTE
A/YYMD DFC 19 YRT 50 = D2_I6YMD;
B/I8YYMD=A;
C/I8YYMD=AYMD(B,1,C);
END
```

Output:

PAGE 1

Start with a computed field called A. It has a pivot year of 1950. Converting it from I6YMD to YYMD sets the century digits to '20' for those years which are less than 50. Create old date field B. Pass B into the AYMD subroutine adding 1 day to the date.

D2_I6YMD	A	B	C
-----	-	-	-
97/09/16	1997/09/16	1997/09/16	1997/09/17
00/02/29	2000/02/29	2000/02/29	2000/03/01
01/02/28	2001/02/28	2001/02/28	2001/03/01
00/02/28	2000/02/28	2000/02/28	2000/02/29

Date Literals Interpretation Table

This table illustrates the behavior of FOCUS date formats. The columns indicate the number of input digits for a date format. The rows indicate the usage or format of the field. The intersection of row and column describes the result of input and format.

	1	2	3	4
YYMD	*	*	CC00/0m/dd	CC00/mm/dd
MDYY	*	*	*	*
DMYY	*	*	*	*
YMD	*	*	CC00/0m/dd	CC00/mm/dd
MDY	*	*	*	*

DMY	*	*	*	*
YYM	CC00/0m	CC00/mm	CC0y/mm	CCyy/mm
MY	*	*	*	*
YM	CC00/0m	CC00/mm	CC0y/mm	CCyy/mm
MY	*	*	0m/CCyy	mm/CCyy
M	0m	mm	*	*
YYQ	CC00/q	CC0y/q	CCyy/q	0yyy/q
QYY	*	*	q/CCyy	*
YQ	CC00/q	CC0y/q	CCyy/q	0yyy/q
QY	*	*	q/CCyy	*
Q	q	*	*	*
JUL	CC00/00d	CC00/0dd	CC00/ddd	CC0y/ddd
YY	000y	00yy	0yyy	yyyy
Y	0y	yy	*	*
D	0d	dd	*	*
W	w	*	*	*

	5	6	7	8
YYMD	CC0y/mm/dd	CCyy/mm/dd	0yyy/mm/dd	yyyy/mm/dd
MDYY	0m/dd/CCyy	mm/dd/Ccyy	0m/dd/yyyy	mm/dd/yyyy
DMYY	0d/mm/CCyy	dd/mm/Ccyy	0d/mm/yyyy	dd/mm/yyyy
YMD	CC0y/mm/dd	CCyy/mm/dd	0yyy/mm/dd	yyyy/mm/dd
MDY	0m/dd/CCyy	mm/dd/Ccyy	0m/dd/yyyy	mm/dd/yyyy
DMY	0d/mm/CCyy	dd/mm/Ccyy	0d/mm/yyyy	dd/mm/yyyy
YYM	0yyy/mm	yyyy/mm	*	*
MYY	0m/yyyy	mm/yyyy	*	*
YM	0yyy/mm	yyyy/mm	*	*
MY	0m/yyyy	mm/yyyy	*	*
M	*	*	*	*
YYQ	yyyy/q	*	*	*
QYY	q/yyyy	*	*	*
YQ	yyyy/q	*	*	*
QY	q/yyyy	*	*	*
Q	*	*	*	*

JUL	CCyy/ddd	*	*	*
YY	*	*	*	*
Y	*	*	*	*
D	*	*	*	*
W	*	*	*	*

- CC stands for two century digits provided by DFC/YRT settings.
- * stands for error message FOC177 (invalid date constant).
- FOCUS reads date literals from right to left.

Reference Special Considerations

You may need to deactivate the new subroutines if you are hardcoding the century digits based on specific years. The windowing technique determines what the century digits should be for each subroutine call.

NF623: Increasing the Number of Verbs in a Report Request

The number of verbs for a multi-verb request, has been increased from 6 to 16.

Usage

This is extremely useful for executing complex reports. The original requirement of having the detail verb last still applies. In other words, PRINT or LIST must be coded last in the list. The aggregation verb (SUM or COUNT) is coded prior to PRINT or LIST.

Syntax

Not applicable

Example Repeating Aggregation of the PCT_INC Field

The following example shows a repeating aggregation of the PCT_INC field out of the EMPLOYEE FOCUS file. This example has 15 verb objects, which are aggregated, and 1 verb object, which lists the detail records for LAST_NAME and FIRST_NAME. The CURR_SAL field is also counted for the total number of records, which passed the IF on DEPARTMENT. The resulting report is an effective illustration of how this feature is used.

```
TABLE FILE EMPLOYEE
SUM PCT_INC
SUM PCT_INC
SUM PCT_INC
SUM PCT_INC
SUM PCT_INC
SUM PCT_INC
SUM PCT_INC
SUM PCT_INC
SUM PCT_INC
SUM PCT_INC
SUM GROSS
SUM DED_AMT
SUM SALARY
SUM ED_HRS
SUM CNT.CURR_SAL BY DEPARTMENT
LIST LAST_NAME FIRST_NAME
BY DEPARTMENT BY EMP_ID
IF DEPARTMENT EQ 'PRODUCTION'
END
```

Special Considerations

Not Applicable

Reference Error Messages

If another SUM PCT_INC is added to the above request a diagnostic displays:

```
(FOC019) THERE ARE TOO MANY VERBS IN THE REQUEST BYPASSING TO END OF
COMMAND
```

The total number of verbs in the request would be 17 which exceeds the new limit. This is the same error message that displays in releases prior to R7.0.8 when the old limit of 6 is exceeded.

NF626: JAVA Graph Wizard

Java Graph Wizard guides a user step-by-step through creating a graph. The Graph Wizard is an alternative to stored graph procedures and generates FOCUS graph syntax from the user's input.

Complete documentation for the Web Interface product can be found in the *Web Interface User's Manual and Installation Guide Release 7.0.8* (DN1001038.1097).

NF628: Automatic Activation of Web Interface for Web Browser Users

Beginning with Release 7.0.8, FOCUS automatically activates the interactive Web environment for FOCUS Web Interface users entering via the Web Interface Server or WEB390.

Usage

FOCUS automates the activation of this environment by internally issuing two SET commands (see “Syntax” below).

Syntax How to Activate the Web Interface

```
SET HTMLMODE=ON  
SET ONLINE-FMT=HTML
```

Examples

N/A - see “Syntax.”

Special Consideration

Sites wishing to activate the Web environment for FOCUS Web Interface users on a selective basis, can do so by turning the SET commands off and reissuing them selectively to activate the environment for only certain users. Complete documentation for the Web Interface product can be found in the Web Interface User’s Manual and Installation Guide Release 7.0.8 (DN1001038.1097).

Error Messages

None

NF630: Querying Which PTFs Have Been Applied for a Specific Release

By issuing a command at the FOCUS prompt, you are able to view a list of ptfs that have been applied to the version of FOCUS you are currently using.

Example Querying the PTFs

```
? ptf
```

Example Using the ? PTF Command

The following displays a screen containing the entering of the ? ptf command, followed by a sample result to the query.

```
FOCUS 7.0.8 10/20/1997 11.44.11 9999.01
>
? ptf
PTFS APPLIED TO RELEASE 7.0.8
FROM PTFABLE LOCATED IN FOCLIB LOADLIB F
```

COUNT	PTF NUM	CREATED	APPLIED
1)	12345	19970403	19971020
2)	12444	19970702	19971020
3)	12499	19970808	19971020

Special Considerations

Not applicable

NF631: Extended Plists

VM FOCUS now issues commands to CMS using CMS's 'Extended Plist'. This will enable FOCUS users to issue CMS commands such as STORMAP and PIPE that require the extended plist. It will also enable FOCUS users to specify FILEDEF, ACCESS and other CMS commands with parameters that are longer than eight characters.

Usage

The standard plist or tokenized plist uppercases each token, left-justifies it, and truncates it to a length of 8 bytes. Extended plist has no limit on the length of the plist passed.

Syntax

N/A

Reference Examples

Without extended plist the command:

```
FILEDEF MINE DSN TEST.SAMPLE.MAY
```

would read as:

```
FILEDEF  
MINE  
DSN  
TEST.SAM
```

In this case the Data Set Name has lost the 'PLE.MAY' part of the qualifier. A tokenized plist is limited to 32 tokens, each of which may be between 1 and 8 bytes. The extended plist consists of a control block that points to an uppercase command token, a pointer to the start of the option list, and a pointer to the end of the option list. The option list is left unchanged. Using extended plist with the previous example, would pass the following:

```
FILEDEF  
MINE DSN TEST.SAMPLE.MAY
```

With extended plist, all of the Data Set Name is preserved.

Reference Special Considerations

FOCUS uses a FILEDEF for SYSIN to read from the terminal and from FOCEXECs. By default FILEDEF uppercases all data. Reissuing FILEDEF with the LOWCASE option, allows a lowercase option list to be passed to VM. This requires all FOCUS commands to be typed in uppercase. This may be used for special cases when a lowercase plist is required

Error Messages

N/A

NF640: Dynamic Language Environment (LE) Support

IBM's recommended platform for high-level language products is known as Language Environment for VM and MVS. It provides a unified platform for runtime services used by LE supported languages. FOCUS user-written subroutines can now be linked using IBM's LE environment. LE support is available for both the MVS and VM operating systems.

This feature enhances our currently announced support for IBM Language Environment. It incorporates the automatic pre-initialization of a Language Environment Enclave or Sub-Enclave as required. This allows customer written HLL subroutines that were linked with LE libraries to run most efficiently. The implementation also allows non-LE-linked subroutines to run as well. FOCUS determines the characteristics of these subroutines and invokes them using LE interface module CEEPIPI if they are LE linked or in the traditional manner if they are not.

Usage

All standard FOCUS subroutines, whether linked prior to LE or using the LE single runtime library are supported depending on Language Environment restrictions. Existing FOCUS subroutine libraries need not be recompiled and relinked unless your site converts to a single run-time library.

Reference Special Considerations

Refer to IBM manual , SC28-1944 *Language Environment for OS/390 and VM Run-Time Migration Guide*, for instructions on how to use this environment and conversion limitations.

NF642: Increased DEFINE Limitation

The limit of 256 for the number of DEFINES allowed in FOCUS has been removed. The limit now is dependent on the amount of memory available. The number of fields, both real and defined that can be referenced in a single request is still 256. It is possible to define as many fields as you need, as long as you have enough memory and the total number of fields referenced in a request is no more than 256.

NF645: WEBHOME

FOCUS 7.0.8 introduces the Web Interface feature 'WEBHOME' which enables Web application developers to specify execution of a default FOCEXEC procedure in situations where FOCUS would normally return to command level. This allows them to prevent application users from accidentally or intentionally accessing command level FOCUS from within a Web application. Previously, FOCUS always displayed the 'FOCUS Interactive' screen by default whenever a Web Interface user entered FOCUS or after they ran a drill-down report or graph that cleared FOCSTACK. With this feature, developers can now display a menu or form that they choose, instead of returning to the FOCUS command level. This will ensure that the FOCUS application environment remains intact and consistent even when Web applications include drill-downs and/or allow users to access the ad hoc Java report and graph generation tools.

Usage

WEBHOME is activated via a FOCUS SET command that you can specify globally for a FOCUS session or issue in a FOCEXEC for a specific application.

Syntax Identifying a FOCEXEC to Run Automatically

```
SET WEBHOME= {focexecname |OFF}
```

where:

focexecname

Is the 1 to 8-character filename of the FOCEXEC to be called instead of returning to the FOCUS command level and displaying the FOCUS Interactive screen

In CMS FOCUS, the focexecname must have a filetype of FOCEXEC.

In MVS FOCUS (TSO or MSO) the FOCEXEC must be a member of a partitioned dataset allocated to ddname FOCEXEC.

OFF

The default, OFF, disables WEBHOME and restores the default behavior - the FOCUS Interactive screen is displayed whenever FOCUS returns to command level.

Example Automatically Running a FOCEXEC

```
SET WEBHOME=APPMENU1
```

The above SET command causes the FOCEXEC named APPMENU1 to be called each time FOCUS would normally return to command level.

Reference Special Consideration

This SET command is valid only in the Web Interface environment. It has no effect in a standard FOCUS 3270 session.

Error Messages

```
This SET command only valid in Web Interface Environment.
```

NF647: Extended Support for Scandinavian External Sort

FOCUS supports external sort with the Scandinavian National Languages Character set, and is able to pass the sort sequences for Swedish , Danish, Finnish, and Norwegian to the external sorting products. To specify the National Language Support Environment, use the LANG parameter as described in Section 21 of the FOCUS 7.0 Users Manual.

Project 2000 - Phase III

The third phase of our year 2000 project for FOCUS includes rewritten user subroutines which perform date manipulations. They are: AYMD, AYM, YM, CHGDAT, GREGDT, JULDAT, DAXxx, DTxxx and FOCUS functions YMD,DMY,MDY. These subroutines and functions will now allow for century digit interpretation via the DEFCENT and YRTHRESH FOCUS settings. Date calculations for these subroutines may now extend beyond year 1999. The subroutines have also been enhanced to respect the last argument which may contain the output format from the subroutine.

Index

Symbols

&DMYY
&FOCERRNUM in Adabas Write
&MDYY
&RETCODE
&YYMD
? FILTER
? FUNCTION command
? PTF
? SET FOR
? SET NOT

Numerics

495 display fields

A

Access File for Adabas Write
 sample
Access File for FOCUS data sources
 attributes
 DATANAME attribute
 introduction
 LOCATION attribute
 MASTERNAME attribute
 syntax
 WHERE attribute
ACCESSFILE attribute

ACCTNAME

ACCTPASS

ACTUAL format for date-time data type

Adabas Data Adapter Enhancements

FOCUS 7.1

NF785 The Adabas Write Data Adapter for FOCUS

Adabas Write Data Adapter

Access File changes

delete rules

error messages

examples

INSERT and UNQKEYNAME

insert rules

Master File changes

non-updatable fields in MAINTAIN

non-updatable fields in MODIFY

options

return codes

sample Access File

sample Master File

synonyms

testing FOCERROR

transaction control

UPDATE rules

user errors

aggregating and sorting report columns

aggregation

and sorting

by External Sort

controlling retrieval order

ALL parameter

and MULTIPATH
CHECK FILE HOLD

ALLOC

For multiple volumes

Allocation

Default space in TABLA
Multiple units
Multi-volume
Of FOCUS files automatically

ALLOWCVTERR

And DATEDISPLAY
And MISSING

assignment of date-time values
attributes for FOCUS Access Files

B

Base dates in FOCUS Reports

Batch

Allocating multi-volume data sources
Pooled Tables

Boundary conditions for Pooled Tables

BUSDAYS (Business day units)

BY TOTAL

Byte precision for COUNT and LIST

C

CA-IDMS Interface

FOCUS 7.0.9

NF584 Dynamically Setting the IDMS DBNAME and DICTNAME

calculated value

sorting by

calling subroutines written in REXX

CALLTYPE

with NEXT

Carriage control

In FORMAT WP files

case logic

CC

CEEPIPI

Changes to the REBUILD Prompt

Changing Retrieval Order with Aggregation

CHECK FILE HOLD ALL

CICS

MSO cooperative processing

CICSCOMM

clear DEFINE functions

CMS

Extended Plists

FILEDEF for creating extract files

Validating userids on sink

CMSOAPI functions

COMBINE

diagram

different SUFFIX

FIND

versus JOIN

comma suppress edit format option

Commands and subpool boundaries

COMMIT WORK

comparison of date-time values

compile REXX subroutines in CMS

Compiler

LE-supported

components for date-time functions

COMPUTE

and currency conversion

and date-time values

sorting by

Console, MSO

Display of IMS PSB

Controlling REBUILD Messages

COUNT, expanding precision for

COUNTWIDTH

CRTFORM for Web390

CS

CURR

Currency

Conversion

Processing

Conversion calculations

Database for conversions

Euro support

Field in a data source

Sample codes

CURRENCY_ID

D

Danish external sort

data source, token delimited

data type, date-time

ACTUAL format

entering values

USAGE

DATANAME attribute

DATASET in a Master File

behavior in fixed-format sequential data sources

behavior in FOCUS data sources

behavior in VSAM data sources

priority in the Master File

syntax for fixed-format data sources

syntax for FOCUS data sources

syntax for VSAM data sources

Date

Adding date units

Base date in FOCUS Reports

Business day units

Converting formats

Converting legacy

Difference between two

Displaying invalid smart dates in reports

Functions for the Year 2000

Handling for the Year 2000 in FOCUS

Holidays

Literal interpretation

MAINTAIN functions

Moving

Subtracting date units

System, altering for testing

Variable, displaying without separators

Weekday units

Date literal interpretation table

date string

formatted string format

numeric string format

translated string format

DATEADD

Using in MAINTAIN

DATECVT

DATEDIF

Using in MAINTAIN

DATEDISPLAY

And ALLOWCVTERR

DATEFORMAT parameter

DATEMOV

Using in MAINTAIN

Date-Time Data Type

ACTUAL format

and HOLD files

and missing values

and SAVE file

assignment

comparison

component names in functions

DATEFORMAT parameter

describing

DTSTRICT parameter

entering values

formatted string date format

functions

HADD function

HCNVRT function

Date-Time Data Type (*continued*)

- HDATE** function
- HDIFF** function
- HDTTM** function
- HGETC** function
- HINPUT** function
- HMIDNT** function
- HNAME** function
- HPART** function
- HSETPT** function
- HTIME** function
- ISO standards**
 - numeric string date format
 - translated string date format
- USAGE**
- WEEKFIRST** parameter

DB2 Interface

- IF-THEN-ELSE** Optimization
- SET ISOLATION**
- SET OPTIFTHENELSE**

DBCS character support in Teradata

DBNAME

DDNAME

- Default space**
- For REXX subroutines**

DEFCENT

- And CHECK FILE HOLD ALL**

DEFINE

- And currency conversion**
- and date-time values**
- Increased limit**

DEFINE functions

clearing

querying

DELETE

rules for Adabas data sources

delimiter

descriptors in Adabas Write

DFSORT

Dialogue Manager TRUNCATE Function

DICTNAME

DIF format in Web Interface

Difference between two dates

display commands

number in a report request

display fields

raised limits

DNS Names Support

DT format for date-time constants

DTSTRICT parameter

DU

dummy segment (SYSTEM99)

DYNAM

Allocating multiple volumes

Support for existing relative GDG numbers

Support for unit count

Dynamic Language Environment support

E

EDA

IMS PSB display in console

EDACFG

embedding text fields in headings

Enhancement to ? SET (FOR, NOT)

error

- messages for Adabas Write

- messages for Pooled Tables

- processing for Pooled Tables

error handling

- Adabas

- ROLLBACK WORK**

- testing with &RETCODE

ERRORRUN

ERRORS, IBITABLA default space allocation table

errors, user

- for Adabas Write

Escape character for LIKE

ESTLINES

ESTRECORDS

Euro support

- Calculations

- Currency conversion

- Processing

- Currency database

- Currency-denominated field

- Sample currency codes

EUROFILE

examples for Adabas Write

Excel (HOLD format)

EXCEL format in Web Interface

EXECLOAD

EXTAGGR

External Sort

Aggregation by

Controlling retrieval order

HOLD from

EXTHOLD

F

FDEFCENT

And CHECK FILE HOLD ALL

FILEDEF for extract files

files, token delimited

FILETYPE for REXX subroutines

Filter

And joins

Assigning to a File for Reporting

Query command

SET command

FIND

Finish external sort

FOC144 message

eliminating with MULTIPATH parameter

FOC2GIGDB

FOCALLOC

FOCCTL.DATA default space allocation table

FOCERROR in Adabas Write

FOCPARM

Configuring Pooled Tables in

FOCPOOLT

FOCPROF - The System Wide Profile

FOCSORT

Allocating multiple units

Allocating multiple volumes

FOCSUACC

FOCUS 7.0.8

NF550 EDA/MSO Console Display for IMS PSB

NF564 Pooled Tables

NF566 MSO/CICS Cooperative Processing

NF568 DB2 Interface IF-THEN-ELSE Optimization

NF571 DB2 Interface SET ISOLATION Command

NF574 System 2000 Interface Trace Facility

NF579 Assigning Screening Conditions to a File for Reporting Purposes

NF583 Teradata Outer Join Optimization

NF586 Expanding Byte Precision for COUNT and LIST

NF594 JAVA Report Assist

NF605 Date Handling for the Year 2000 in FOCUS

NF607 Default Space Allocation Table for Work Files

NF607 TABLA Enhancements

NF609 Sink Validation of Userids in CMS

NF617 Automatic Allocation of FOCUS Files

NF619 -HTMLFORM SAVE

NF620 Year 2000 Subroutines

NF623 Increasing the Number of Verbs in a Report Request

NF626 JAVA Graph Wizard

NF628 Automatic Activation of Web Interface

NF630 Querying Which PTFs Have Been Applied for a Specific Release

NF631 Extended Plists

NF640 Dynamic Language Environment (LE) Support

NF642 Increased DEFINE Limitation

NF645 WEBHOME

NF647 Extended Support for Scandinavian External Sort

FOCUS 7.0.8R

NF557 REBUILD Enhancement - Legacy Date Conversion
NF653 Displaying Base Dates in FOCUS Reports
NF659 CHECK FILE HOLD ALL
NF700 New Date Math Functions for Year 2000
NF703 Displaying Invalid Smart Dates in Reports
NF705 Enhancement to YRTHRESH Command
NF708 Enhancement to the TODAY subroutine
NF709 Displaying a Date Variable Without Separators
NF710 Field FORMAT=YYJUL
NF711 Altering Your System Date for Testing
NF713 MSO Log Changes
NF714 LE Support

FOCUS 7.0.9

NF575 Fusion
NF584 Dynamically Setting the IDMS DBNAME and DICTNAME
NF597 Aggregation by External Sort
NF652 Teradata Interface Kanji Support
NF654 HOLD From External Sort
NF655 FOCPROF - The System Wide Profile
NF656 Controlling REBUILD Messages
NF660 Multi-volume Support in MVS FOCUS
NF670 DYNAM Support for Unit Count
NF673 Model 204 Interface Account Split
NF683 Web Interface support for Maintain Winforms
NF684 PCHOLD for Non-Html Files
NF691 Escape Character for LIKE
NF716 Euro Currency Support
NF718 DYNAM Support for Existing Relative GDG Numbers
NF720 SQLJOIN OUTER Setting for Relational Interfaces
NF722 FOCUS Client DNS Names Support

FOCUS 7.0.9 (continued)

- NF728 Changing Retrieval Order with Aggregation
- NF730 Hold Format PDF
- NF735 Enhancement to ? SET
- NF740 Changes to the REBUILD PROMPT
- NF744 HOLD FORMAT EXCEL
- NF745 ? PTF Enhancements
- NF746 Leading Zeros
- NF748 HOLD FORMAT WP With Carriage Control

FOCUS 7.1

- NF692 Aggregating and Sorting Report Columns
- NF696 Calling Subroutines Written in REXX
- NF731 Reporting From Independent Paths
- NF749 HOLD FORMAT INTERNAL
- NF750 DATASET in a Master File
- NF751 Date-Time Data Type
- NF755 Using FILEDEF for Creating Extract Files
- NF759 Increased Number of Display Fields
- NF761 Comma Suppress Edit Format Option
- NF762 Percent Edit Format Option
- NF766 DEFINE Functions
- NF773 Token Delimited Files
- NF777 Partitioned FOCUS data sources
- NF777 Two-Gigabyte FOCUS Database Support
- NF778 Dialogue Manager TRUNCATE Function
- NF779 FOCUS CRTFORM HTML Translation
- NF781 Embedding Text Fields in Headings
- NF782 Oracle Data Adapter IXSPACE Setting
- NF785 Adabas Write Data Adapter for FOCUS

FOCUS Client

FOCUS 7.0.9

NF722 DNS Names Support

FOCUS CRTFORM HTML Translation

FOCUS database

Access File

allocating Multiple units

allocating multiple volumes

partitioned

two-gigabyte

FOCUS referential integrity

DELETE

INCLUDE

format

date-time

ACTUAL

overrides

YYJUL

formatted string date format

FSTRACE

For System 2000 Interface

Functional Area

Adabas Data Adapter

NF785 The Adabas Write Data Adapter for FOCUS

CA-IDMS Interface

NF584 Dynamically Setting the IDMS DBNAME and DICTNAME

FOCUS Client

NF722 FOCUS Client DNS Names Support

Fusion

NF575 Fusion

Functional Area (*continued*)

General Enhancements

- NF607 TABLA Enhancements (Default Space Allocation Table for Work Files)
- NF609 Sink Validation of Userids in CMS
- NF630 Querying Which PTFs Have Been Applied for a Specific Release
- NF631 Extended Plists
- NF640 Dynamic Language Environment (LE) Support
- NF655 FOCPROF - The System Wide Profile
- NF656 Controlling REBUILD Messages
- NF670 DYNAM Support for Unit Count
- NF696 Calling Subroutines Written in REXX
- NF714 LE Support
- NF718 DYNAM Support for Existing Relative GDG Numbers
- NF735 Enhancement to ? SET
- NF740 Changes to the REBUILD Prompt
- NF745 ? PTF Enhancements
- NF746 Leading Zeros
- NF750 DATASET in a Master File
- NF751 Date-Time Data Type
- NF773 Token Delimited Files
- NF777 Two-Gigabyte FOCUS Database Support
- NF778 Dialogue Manager TRUNCATE Function
- NF779 FOCUS CRTFORM HTML Translation

IMS Interface

- NF550 EDA/MSO Console Display for IMS PSB

Model 204 Interface

- NF572 Invisible Ordered Character and Ordered Numeric Data Type Key Support
- NF673 Model 204 Interface Account Split

Functional Area (*continued*)

Multi-Session Option

NF466 MSO/CICS Cooperative Processing

National Language Support

NF647 Extended Support for Scandinavian External Sort

Oracle Data Adapter

NF782 Oracle Data Adapter IXSPACE Setting

Performance Enhancements

NF564 Pooled Tables

NF597 Aggregation by External Sort

NF617 Automatic Allocation of FOCUS Files

NF654 HOLD From External Sort

NF728 Changing Retrieval Order with Aggregation

NF777 Partitioned FOCUS Data Sources

Raised Limits

NF642 Increased DEFINE Limitation

NF759 Increased Number of Display Fields

NF777 Two-Gigabyte FOCUS Database Support

Relational Interfaces

NF568 DB2 Interface IF-THEN-ELSE Optimization

NF571 DB2 Interface SET ISOLATION Command

NF583 Teradata Outer Join Optimization

NF720 SQLJOIN OUTER Setting

Functional Area (*continued*)

Reporting Enhancements

- NF579 Assigning Screening Conditions to a File for Reporting Purposes
- NF586 Expanding Byte Precision for COUNT and LIST
- NF623 Increasing the Number of Verbs in a Report Request
- NF691 Escape Character for LIKE
- NF692 Aggregating and Sorting Report Columns
- NF731 Reporting From Independent Paths
- NF744 HOLD FORMAT EXCEL
- NF748 HOLD FORMAT WP with Carriage Control
- NF749 HOLD FORMAT INTERNAL
- NF755 Using FILEDEF for Creating Extract Files
- NF759 Increased Number of Display Fields
- NF761 Comma Suppress Edit Format Option
- NF762 Percent Edit Format Option
- NF766 DEFINE Functions
- NF781 Embedding Text Fields in Headings

System 2000 Interface

- NF574 System 2000 Interface Trace Facility

Teradata Interface

- NF583 Teradata Outer Join Optimization
- NF652 Kanji support

Web Interface for FOCUS

- NF594 JAVA Report Assist
- NF619 -HTMLFORM SAVE
- NF626 JAVA Graph Wizard
- NF628 Automatic Activation of Web Interface
- NF645 WEBHOME
- NF683 Web Interface support for Maintain Winforms
- NF684 PCHOLD for Non-Html Files
- NF730 Hold Format PDF

Functional Area (*continued*)

Year 2000 Enhancements

NF557 REBUILD Enhancement - Legacy Date Conversion
NF605 Date Handling for the Year 2000 in FOCUS
NF620 Year 2000 Subroutines
NF653 Displaying Base Dates in FOCUS Reports
NF659 CHECK FILE HOLD ALL
NF700 New Date Math Functions for the Year 2000
NF703 Displaying Invalid Smart Dates in Reports
NF705 YRTHRESH As an offset from current year
NF708 Enhancement to the TODAY Subroutine
NF709 Displaying a Date Variable Without Separators
NF710 Field FORMAT=YYJUL
NF711 Altering Your System Date For Testing
NF713 MSO Log Changes

functions

clearing
date-time
date-time component names
DEFINE
FOCUS
For the Year 2000
HADD
HCNVRT
HDATE
HDIFF
HDTTM
HGETC
HINPUT
HMIDNT
HNAME

functions (*continued*)

HPART

HSETPT

HTIME

querying

Fusion

FUSREXX macro

CMS FILETYPE

compiling in CMS

date return value

integer return value

loading in CMS

multiple tokens in parameters

MVS DDNAME

parameters

search order

FYRTHRESH

And **CHECK FILE HOLD ALL**

G

GDG

DYNAM support for existing relative numbers

General Enhancements

FOCUS 7.0.8

NF607 TABLA Enhancements (Default Space Allocation Table for Work Files)

NF609 Sink Validation of Userids in CMS

NF630 Querying Which PTFs Have Been Applied for a Specific Release

NF631 Extended Plists

NF640 Dynamic Language Environment (LE) Support

General Enhancements (*continued*)

FOCUS 7.0.8R

NF714 LE Support

FOCUS 7.0.9

NF655 FOCPROF - The System Wide Profile

NF656 Controlling REBUILD Messages

NF670 DYNAM Support for Unit Count

NF718 DYNAM Support for Existing Relative GDG Numbers

NF735 Enhancement to ? SET

NF740 Changes to the REBUILD Prompt

NF745 ? PTF Enhancements

NF746 Leading Zeros

FOCUS 7.1

NF696 Calling Subroutines Written in REXX

NF750 DATASET in a Master File

NF751 Date-Time Data Type

NF773 Token Delimited Files

NF777 Two-Gigabyte FOCUS Database Support

NF778 Dialogue Manager TRUNCATE Function

NF779 FOCUS CRTFORM HTML Translation

Graph Wizard, JAVA

H

H format code

H USAGE format

HADD date-time function

HCNVRT date-time function

HDATE date-time function

HDAY

HDIFF date-time function

HDTTM date-time function

headings, embedding text in

HGETC date-time function

HINPUT date-time function

HLMAIN

HMIDNT date-time function

HNAME date-time function

HOLD

ALL

allocating with **FILEDEF**

and date-time data type

And **DEFCENT**, **YRTHRESH**

FORMAT EXCEL

FORMAT INTERNAL

Format **PDF**

FORMAT WP

With carriage control

From external sort

Holidays

HPART date-time function

HSETPT date-time function

HTIME date-time function

-HTMLFORM SAVE

I

IBITABLA

Default space allocation

IBMLE

Recommended settings

IF

and date-time values

Optimization in DB2 Interface

IMS Interface

FOCUS 7.0.8

NF550 EDA/MSO Console Display for IMS PSB

INCLUDE

increased number of display fields

independent paths

and FOC144 message

and SET ALL

required segments

resource requirements for MULTIPATH

index in MODIFY

index space parameters for Oracle

INSERT

rules for Adabas data sources

Installation

Pooled Tables

All systems

MVS

VM/CMS

integer return value from REXX subroutine

intelligent partitioning

Invisible Ordered Character and Ordered Numeric Data Type Key Support

ISO standards for date-time

Isolation, setting level in DB2 Interface

IXSPACE

J

JAVA Graph Wizard

JAVA Report Assist

JOIN

LOOKUP

versus **COMBINE**

joins

and **Access Files**

and **filters**

and **partitioned FOCUS data sources**

left outer

controlling

outer in Teradata Interface

K

Kanji character set

Key, invisible Ordered Character and Ordered Numeric support

L

Language Environment (LE) Support

In FOCUS 7.0.8

In FOCUS 7.0.8R

LE support

Languages

Recommended settings

LEADZERO

LIKE with escape character

limits

data sources per MAINTAIN request

data sources per MODIFY request

DEFINE

FOCUS database size

number of display fields

LIST

Expanding precision for

LOCATION attribute

in Access File

logical unit of work

Logs, MSO, support for four-digit years

LOOKUP

LOTUS format in Web Interface

M

MAINTAIN

Adabas data sources

Date math functions

DELETE

INCLUDE

limits

MATCH

NEXT

non-updatable Adabas fields

prerequisites

processing overview

referential integrity

transaction control for Adabas Write

UPDATE

MAINTAIN (*continued*)

Winforms in Web Interface
without unique keys
maintaining data sources

Master File

ACCESSFILE attribute
for Adabas Write
sample for Adabas Write

MASTERNAME attribute

MATCH

DELETE
INCLUDE
MAINTAIN
UPDATE

MAXEXTSRTS

memory management with Pooled Tables
messages

For Adabas data source
Pooled Tables

MISSING

And ALLOWCVTERR
and date-time data type

Model 204 Interface

FOCUS 7.0.8
NF572 Invisible Ordered Character and Ordered Numeric key support

FOCUS 7.0.9
NF673 Model 204 Interface Account Split

IOA and ION

MODIFY

- Adabas data sources
- case logic
- COMBINE
- COMBINE SUFFIX
- COMBINE versus JOIN
- COMMIT WORK
- DELETE
- differences from standard FOCUS
- FIND
- INCLUDE
- limits
- LOOKUP
- MATCH
- NEXT
- non-updatable Adabas fields
- prerequisites
- processing overview
- referential integrity
- ROLLBACK WORK
- transaction control for Adabas Write
- UPDATE
- VALIDATE
- without unique keys

MSO

- Allocating multiple units
- Allocating multiple volumes
- CICS cooperative processing
- FOCUS 7.0.8
 - NF466 MSO/CICS Cooperative Processing**
- IMS PSB display in console
- Logs with four-digit years

MULTIPATH parameter

and FOC144 message

and SET ALL

required segments

resource requirements

Multiple

Units in MVS FOCUS

Volumes

Default allocations

Multi-verb request

Number of verbs in

MVS

LE support

MVS batch

Allocating multi-volume data sources

N

National Language Support

FOCUS 7.0.8

NF647 Extended Support for Scandinavian External Sort

Scandinavian external sort

NC

New date

Displaying invalid dates in reports

New Features

FOCUS 7.0.8

Assigning Screening Conditions to a File for Reporting Purposes
Automatic Activation of Web Interface
Automatic Allocation of FOCUS Files
Date Handling for the Year 2000 in FOCUS
DB2 Interface IF-THEN-ELSE Optimization
DB2 Interface SET ISOLATION Command
Default Space Allocation Table for Work Files
Dynamic Language Environment (LE) Support
EDA/MSO Console Display for IMS PSB
Expanding Byte Precision for COUNT and LIST
Extended Plists
Extended Support for Scandinavian External Sort
-HTMLFORM SAVE
Increased DEFINE Limitation
Increasing the Number of Verbs in a Report Request
Invisible Ordered Character and Ordered Numeric Data Type Key Support
JAVA Graph Wizard
JAVA Report Assist
MSO/CICS Cooperative Processing
Pooled Tables
Querying Which PTFs Have Been Applied for a Specific Release
Sink Validation of Userids in CMS
System 2000 Interface Trace Facility
TABLA Enhancements
Teradata Outer Join Optimization
WEBHOME
Year 2000 Subroutines

New Features (*continued*)

FOCUS 7.0.8R

Altering Your System Date for Testing
CHECK FILE HOLD ALL
Displaying a Date Variable Without Separators
Displaying Base Dates in FOCUS Reports
Displaying Invalid Smart Dates in Reports
Enhancement to the TODAY Subroutine
Enhancement to YRTHRESH Command
Field FORMAT=YYJUL
LE Support
MSO Log Changes
New Date Math Functions for Year 2000
REBUILD Enhancement - Legacy Date Conversion

FOCUS 7.0.9

? PTF Enhancements
Aggregation by External Sort
Changes to the REBUILD PROMPT
Changing Retrieval Order with Aggregation
Controlling REBUILD Messages
DYNAM Support for Existing Relative GDG Numbers
DYNAM Support for Unit Count
Dynamically Setting the IDMS DBNAME and DICTNAME
Enhancement to ? SET
Escape Character for LIKE
Euro Currency Support
FOCPROF - The System Wide Profile
FOCUS Client DNS Names Support
Fusion
HOLD FORMAT EXCEL
Hold Format PDF

New Features

FOCUS 7.0.9 *(continued)*

- HOLD FORMAT WP** With Carriage Control
- HOLD** From External Sort
- Leading Zeros
- Model 204 Interface Account Split
- Multi-volume Support in MVS FOCUS
- PCHOLD** for Non-Html Files
- SQLJOIN OUTER** Setting for Relational Interfaces
- Teradata Interface Kanji Support
- Web Interface Support for Maintain Winforms

FOCUS 7.1

- Aggregating and Sorting Report Columns
- Calling Subroutines Written in REXX
- Comma Suppress Edit Format Option
- DATASET** in a Master File
- Date-Time Data Type
- DEFINE** Functions
- Dialogue Manager **TRUNCATE** Function
- Embedding Text Fields in Headings
- FOCUS CRTFORM HTML** Translation
- HOLD FORMAT INTERNAL**
- Increased Number of Display Fields
- Oracle Data Adapter **IXSPACE** Setting
- Partitioned **FOCUS** Data Sources
- Percent Edit Format Option
- Reporting From Independent Paths
- The Adabas Write Data Adapter for **FOCUS**
- Token Delimited Files
- Two-Gigabyte **FOCUS** Database Support
- Using **FILEDEF** for Creating Extract Files

NEXT

MAINTAIN

non-unique key in Adabas Write

unique key in Adabas Write

without MATCH

NLS

Scandinavian external sort

NOCC

Norwegian external sort

Numeric string date format

O

OPTIFTHENELSE

Optimization

DB2 Interface IF-THEN-ELSE

Of outer joins

Controlling

Teradata Interface outer join

options

for Adabas Write

Oracle Data Adapter Enhancements

FOCUS 7.1

NF782 Oracle Data Adapter IXSPACE Setting

Oracle Data Adapter IXSPACE Setting

Outer Join

Optimization

Controlling

P

padding, preventing in HOLD files

partitioned FOCUS data sources

partitions

and joins

FOCUS database

intelligent

paths, independent

and FOC144 message

and SET ALL

required segments

resource requirements for MULTIPATH

PCHOLD for Non-Html Files

PDF format

Creating

In Web Interface

percent edit format option

Performance Enhancements

FOCUS 7.0.8

NF564 Pooled Tables

NF617 Automatic Allocation of FOCUS Files

FOCUS 7.0.9

NF597 Aggregation by External Sort

NF654 HOLD From External Sort

NF728 Changing Retrieval Order with Aggregation

FOCUS 7.1

NF777 Partitioned FOCUS data sources

Plists, extended

Pooled Tables

Additional information

And common selection criteria

And non-relational databases

And relational databases

Batch mode

Commands and subpool boundaries

Configuring in FOCPARM

Error processing

Example

FOCPARM file

FOCPOOLT

Installing

All systems

MVS

VM/CMS

Memory management

Memory needs

Messages

Questions about

Single TABLE clusters

Size estimates

Sort selection

Statistics

Subpool boundaries

Subroutines for use with

Temporary work file

Trace facility

Tuning techniques

Precision, expanding for COUNT and LIST

PSB, display in EDA/MSO console

PTF query command

PTF, querying which have been applied

Q

query commands

? FUNCTION

R

Raised Limits

FOCUS 7.0.8

NF642 Increased DEFINE Limitation

FOCUS 7.1

NF759 Increased Number of Display Fields

NF777 Two-Gigabyte FOCUS Database Support

REBUILD

Converting legacy dates

DATE NEW

REBUILDMSG

RECFM VBA

referential integrity

and COMBINE

FOCUS DELETE

FOCUS INCLUDE

inhibiting

Relational Interfaces

FOCUS 7.0.8

NF568 DB2 Interface IF-THEN-ELSE Optimization

NF571 DB2 Interface SET ISOLATION Command

NF583 Teradata Outer Join Optimization

FOCUS 7.0.9

NF720 SQLJOIN OUTER Setting

FOCUS 7.1

NF782 Oracle Data Adapter IXSPACE Setting

Release

Querying which PTFs have been applied for
report

Displaying invalid smart dates in
number of display fields in

Number of verbs in

Pooling requests

Report Assist, JAVA

report headings

embedding text in

text alignment

Reporting Enhancements

FOCUS 7.0.8

NF579 Assigning Screening Conditions to a File for Reporting Purposes

NF586 Expanding Byte Precision for COUNT and LIST

NF623 Increasing the Number of Verbs in a Report Request

FOCUS 7.0.9

NF691 Escape Character for LIKE

NF744 HOLD FORMAT EXCEL

NF748 HOLD FORMAT WP With Carriage Control

Reporting Enhancements (*continued*)

FOCUS 7.1

NF692 Aggregating and Sorting Report Columns

NF731 Reporting From Independent Paths

NF749 HOLD FORMAT INTERNAL

NF755 Using FILEDEF for Creating Extract Files

NF759 Increased Number of Display Fields

NF761 Comma Suppress Edit Format Option

NF762 Percent Edit Format Option

NF766 DEFINE Functions

NF781 Embedding Text Fields in Headings

reporting from independent paths

REPOSITION

requests

number of display fields in

number of verbs in

pooling multiple reports

REXX user-written subroutines

CMS FILETYPE

compiling in CMS

date return value

integer return value

loading in CMS

multiple tokens in parameters

MVS DDNAME

parameters

search order

ROLLBACK WORK

RR

rules

- for delete from Adabas data sources
- for update of Adabas data sources
- insert for Adabas data sources

S

SAVE files

- and date-time data type

Scandinavian National Language character set

- And external sort

Screening conditions

- Assigning to a File for Reporting

segment **SYSTEM99**

segments, required for report

SEGTYPE for Adabas Write

SET

- ACCTNAME**

- ACCTPASS**

- ALL** and **MULTIPATH**

- ALLOWCVTERR**

- BUSDAYS**

- COUNTWIDTH**

- DATEDISPLAY**

- DATEFORMAT**

- DBNAME**

- DICTNAME**

- DTSTRICT**

- ERRORRUN**

- ESTLINES**

- ESTRECORDS**

SET (continued)

EUROFILE

EXTAGGR

EXTHOLD

FILTER

FOC2GIGDB

FOCALLOC

HDAY

IBMLE

ISOLATION in DB2 Interface

IXSPACE (Oracle)

LEADZERO

MAXEXTSRTS

MULTIPATH

OPTIFTHENELSE

POOL

POOLBATCH

POOLFEATURE

POOLMEMORY

POOLRESERVE

SORTLIB

SQLJOIN OUTER

SUMPREFIX

TESTDATE

TRACEOFF

TRACEON

WEBHOME

WEBTAB

WEEKFIRST

Sink

Validating userids in CMS

Smart date

Displaying invalid dates in reports

Sort

Controlling retrieval order

Scandinavian National Language character set

Selection with Pooled Tables

sorting

by a calculated value

Space

Default allocation table in TABLA

SQL

COMMIT WORK

ROLLBACK WORK

SET IXSPACE (Oracle)

SQLJOIN OUTER

subheadings, embedding text in

Subpool

Boundaries and commands

Boundaries for Pooled Tables

Subroutines

And Pooled Tables

And Year 2000

Date literal interpretation table

LE support

Subroutines (*continued*)

REXX

CMS FILETYPE
compiling in CMS
date return value
integer return value
loading in CMS
multiple tokens in parameters
MVS DDNAME
parameters
search order

Year 2000 support

DATEADD
DATECVT
DATEDIFF
DATEMOV
In **MAINTAIN**
TODAY

SUFFIX

COMBINE

SUMPREFIX

Swedish external sort

SYNCSORT

synonyms in Adabas Write

System 2000 Interface

FOCUS 7.0.8

NF574 System 2000 Trace Facility

SYSTEM99

T

TABLA

Default space allocation table

TABLE

Pooling requests

Teradata Interface

FOCUS 7.0.8

NF583 Teradata Outer Join Optimization

FOCUS 7.0.9

NF652 Kanji Support

Outer Join Optimization

TESTDATE

text fields

alignment in headings

embedding in headings

TODAY

And Year 2000

token delimited files

Trace

For Pooled Tables

For System 2000 Interface

transaction control in Adabas Write

COMMIT WORK

ROLLBACK WORK

translated string date format

TRUNCATE

TSO

Allocating multiple units

Allocating multiple volumes

Two-Gigabyte FOCUS Database Support

U

UCOUNT

Allocating multiple units

unique key

and INSERT for Adabas data sources
in Adabas Write

unit of work

Units

Allocating multiple

Business day

Weekday

UNQKEYNAME

and INSERT for Adabas data source

UPDATE

rules for Adabas data sources

UR

USAGE, date-time (H)

user errors for Adabas Write

Userid

Validating by Sink in CMS

using FILEDEF for creating extract files

V

VALIDATE

LOOKUP

Validation

Of userids by Sink in CMS

VARGRAPHIC

Variable

Date without separators

VBA

Verbs

Number in a report request
virtual segment SYSTEM99

VM

CMS extended Plists
FILEDEF for creating extract files
LE support

VMSORT

VOLSER

Multiple

Volume

Allocating multiple

W

Warning messages

Pooled Tables

Web Interface for FOCUS

FOCUS 7.0.8

NF594 JAVA Report Assist
NF619 -HTMLFORM SAVE
NF626 JAVA Graph Wizard
NF628 Automatic Activation of Web Interface
NF645 WEBHOME

FOCUS 7.0.9

NF683 Support for Maintain Winforms
NF684 PCHOLD for Non-Html Files
NF730 Hold Format PDF

WEBHOME

WEBTAB parameter

Weekday units

WEEKFIRST parameter

WHERE test

and date-time values

in Access File

Winforms

In Web Interface

Work files

Default Space Allocation Table

WP format

In Web Interface

With carriage control

write

Access File for Adabas

component for data source maintenance

Master File for Adabas

options for Adabas

Y

Year 2000 Enhancements

FOCUS 7.0.8

NF605 Date handling for the Year 2000 in FOCUS

NF620 Year 2000 Subroutines

FOCUS 7.0.8R

NF557 Converting legacy dates

NF653 Displaying Base Dates in FOCUS Reports

NF659 CHECK FILE HOLD ALL

NF700 New Date Math Functions for the Year 2000

NF703 Displaying invalid smart dates in reports

NF705 YRTHRESH As an offset from current year

NF708 Enhancement to the TODAY Subroutine

NF709 Displaying a Date Variable Without Separators

NF710 Field FORMAT=YYJUL

NF711 Altering Your System Date For Testing

NF713 MSO Log Changes

YRTHRESH

And CHECK FILE HOLD ALL

As an offset from the current year

YYJUL