Information Builders®

FOCUS® for S/390®

Creating Reports
Version 7.0

DN1001056.0599

Contents

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

Printed in the U.S.A.

Preface

This documentation set describes how to use FOCUS® Version 7.0. The documentation set consists of the following components:

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

The users' documentation for FOCUS Version 7.0 is organized to provide you with a useful, comprehensive guide to FOCUS.

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

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

Audience

This manual is intended for any FOCUS user who will access corporate data to produce reports.

How This Manual Is Organized

This manual includes the following chapters:

Chapter/Appendix		**Contents**
1	*Creating Tabular Reports*	Provides an introduction to the TABLE command, a powerful tool for analyzing data.
2	*Displaying Report Data*	Describes ways to retrieve field values from a database and display them.
3	*Viewing and Printing Report Output*	Describes the HotScreen facility for viewing report output.
4	*Sorting Tabular Reports*	Describes how to display report information grouped in a particular order by sorting.
5	*Selecting Records for Your Report*	Describes how to use and specify selection criteria to display only the field values that meet your needs.
6	*Creating Temporary Fields*	Describes how to use the DEFINE and COMPUTE commands to create temporary fields.
7	*Including Totals and Subtotals in Your Report*	Describes how to use subtotals and grand totals to summarize numeric information and aid in interpreting detailed information in a report.
8	*Using Expressions*	Describes how to combine operators, fieldnames, and constants in an expression to derive new values.
9	*Using Functions and Subroutines*	Describes the functions and subroutines provided by FOCUS for performing calculations and data manipulation.
10	*Customizing Tabular Reports*	Describes how to override the default report formats to meet your individual presentation needs.
11	*Styling Reports: StyleSheets*	Describes how to visually style your reports with StyleSheets, used to control report output to be printed on a PostScript printer.
12	*Saving and Reusing Your Report Output*	Describes how to save report output in a wide variety of formats.
13	*Handling Records With Missing Field Values*	Describes how missing data affects report results and how to treat and represent it.

Chapter/Appendix		**Contents**
14	*Joining Data Files*	Describes how to join two or more related data sources to create a larger integrated data structure from which you can report.
15	*Merging Data Files*	Describes how to merge and concatenate two or more data sources into a new permanent data file.
16	*Improving Data Retrieval*	Describes methods of increasing data retrieval and reporting efficiency.
17	*Creating Extended Matrix Reports*	Describes Extended Matrix Reporting (EMR), used to create and present financially oriented data, using inter-row calculations.
18	*Creating Free-Form Reports*	Describes how to present data in an unrestricted (non-tabular) format.
19	*Creating Graphs: GRAPH*	Describes the FOCUS GRAPH facility, which you can use to display data in graph format instead of tabular format.
20	*Using SQL to Create Reports*	Describes how to use SQL to retrieve and analyze FOCUS and RDBMS data.
A	*Master Files and Diagrams*	Contains Master Files and diagrams of sample databases used in the documentation examples.
B	*Error Messages*	Describes how to access FOCUS error messages.
C	*Syntax Summary*	Summarizes FOCUS Table commands and options.
D	*Creating Your Own Subroutines*	Describes how to write subroutines that can be called from FOCUS.
E	*Character Charts*	Lists EBCDIC codes and their corresponding character representations.

Summary of New Features

The FOCUS features and enhancements described in this manual are listed in the following table.

New Feature	**Release/ Version**	**Chapter**
HOLD FORMAT EXCEL HOLD FORMAT WP [CC|NOCC]	7.0.9	Chapter 12, *Saving and Reusing Your Report Output*.
Escape Character for LIKE	7.0.9	Chapter 5, *Selecting Records for Your Report*.
AUTOINDEX on External Index	7.0.9	Chapter 16, *Improving Data Retrieval*.
External Sort to Produce HOLD File	7.0.9	Chapter 4, *Sorting Tabular Reports*.
Aggregation by External Sort	7.0.9	Chapter 4, *Sorting Tabular Reports*.
Changing Retrieval Order With Aggregation	7.0.9	Chapter 4, *Sorting Tabular Reports*.
New Date Math Functions for the Year 2000	7.0.8R	Chapter 9, *Using Functions and Subroutines*.
Enhancement to the TODAY Subroutine	7.0.8R	Chapter 9, *Using Functions and Subroutines*.
SET IBMLE ON/OFF	7.0.8R	Chapter 9, *Using Functions and Subroutines*.
Assigning Screening Conditions to a File for Reporting Purposes	7.0.8	Chapter 5, *Selecting Records for Your Report*.
Expanding Byte Precision for COUNT and LIST	7.0.8	Chapter 2, *Displaying Report Data*.
Year 2000 Subroutines	7.0.8	Chapter 9, *Using Functions and Subroutines*.
Increasing the Number of Verbs in a Report Request	7.0.8	Chapter 4, *Sorting Tabular Reports*.
Dynamic Language Environment (LE) Support	7.0.8	Chapter 9, *Using Functions and Subroutines*.
Increased DEFINE Limitation	7.0.8	Chapter 6, *Creating Temporary Fields*.
Extended Support for Scandinavian External Sort	7.0.8	Chapter 4, *Sorting Tabular Reports*.

New Feature	**Release/ Version**	**Chapter**
PRINTPLUS	7.0.6	Chapter 3, *Viewing and Printing Report Output*.
Enhancements to JOIN	7.0.6	Chapter 14, *Joining Data Files.*
Interpreting Quotation Marks Within Quote-Delimited Literal Strings	7.0.6	Chapter 8, *Using Expressions*.
Estimating SORTWORK Sizes for External Sort	7.0.6	Chapter 4, *Sorting Tabular Reports*.
Keyed Retrievals From FOCUS Extract Files	7.0.5	Chapter 12, *Saving and Reusing Your Report Output.*
Scrolling Report Headings in HotScreen (BYSCROLL)	7.0.5	Chapter 3, *Viewing and Printing Report Output*.
Distinct Operator (DST.)	7.0.3	Chapter 3, *Viewing and Printing Report Output*.

Documentation Conventions

The following conventions apply throughout this manual:

Convention	**Description**
`THIS TYPEFACE`	Denotes a command that you must enter in uppercase, exactly as shown.
`this typeface`	Denotes a value that you must supply.
`{ }`	Indicates two choices from which you must choose one. You type one of these choices, not the braces.
`|`	Separates two mutually exclusive choices in a syntax line. Type one of these choices, not the symbol.
`[ ]`	Indicates optional parameters. None of them is required, but you may select one of them. Type only the information within the brackets, not the brackets.
`underscore`	Indicates the default value.
`...`	Indicates that you can enter a parameter multiple times. Type only the information, not the ellipsis points.
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

See the Information Builders Publications Catalog for the most up-to-date listing and prices of technical publications, plus ordering information. To obtain a catalog, contact the Publications Order Department at (800) 969-4636.

You can also visit our World Wide Web site, http://www.ibi.com, to view a current listing of our publications and to place an order.

Information Builders Systems Journal

The *Information Builders Systems Journal* is a unique technical publication dedicated to providing you with the latest information necessary to enhance your use of FOCUS and all other Information Builders products.

Through its detailed articles, illustrated with code, screen shots, and other visuals, the Journal challenges you to develop better reporting habits, customize features to enhance your systems applications, utilize its tips and techniques for better performance and productivity, and so much more.

You can order the *Information Builders Systems Journal* from the Publications Catalog or from our World Wide Web site, http://www.ibi.com.

Customer Support

Do you have questions about FOCUS?

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

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

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

Information You Should Have

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

- Your six digit site code number (xxxx.xx).
- The FOCEXEC procedure (preferably with line numbers).
- Master File with picture (provided by CHECK FILE).
- Run sheet (beginning at login, including call to FOCUS), containing the following information:
    - ? RELEASE
    - ? FDT
    - ? LET
    - ? LOAD
    - ? COMBINE
    - ? JOIN
    - ? DEFINE
    - ? STAT
    - ? SET/? SET GRAPH
    - ? USE
    - ? TSO DDNAME OR CMS FILEDEF
- The exact nature of the problem:
    - Are the results or the format incorrect; are the text or calculations missing or misplaced?
    - The error message and code, if applicable.
    - Is this related to any other problem?
- Has the procedure or query ever worked in its present form? Has it been changed recently? How often does the problem occur?
- What release of the operating system are you using? Has it, FOCUS, your security system, or an interface system changed?

- Is this problem reproducible? If so, how?
- Have you tried to reproduce your problem in the simplest form possible? For example, if you are having problems joining two databases, have you tried executing a query containing just the code to access the database?
- Do you have a trace file?
- How is the problem affecting your business? Is it halting development or production? Do you just have questions about functionality or documentation?

User Feedback

In an effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual. Please use the Reader Comments form at the end of this manual to relay suggestions for improving the publication or to alert us to corrections. You can also use the Document Enhancement Request Form on our Web site, http://www.ibi.com.

Thank you, in advance, for your comments.

Information Builders Consulting and Training

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

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

1 Creating Tabular Reports

Topics:

- Before You Begin: Requirements
- Developing Your Report Request
- Beginning, Ending, and Issuing a Report Request
- The Parts of a Report Request
- Referring to Fields
- Specifying Columns

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

The simplest form of report that you can produce using the FOCUS reporting language is a columnar report—that is, a report whose information is arranged vertically in columns. This is the basic FOCUS report format, incorporating the fundamental reporting concepts and command syntax. Most of the other report formats build on these concepts and syntax. This chapter describes how to create columnar reports.

Most of the examples in this chapter use the EMPLOYEE sample database. You can see a description of EMPLOYEE in Appendix A, *Master Files and Diagrams*.

If you are new to FOCUS reporting—and especially if you are new to reporting in general—review the following topics before you create your first report.

- **Prerequisites.** You need to access a database and know what information is in it before you can report from it.
- **Development environments.** You develop a report request using an editor and save the request in a stored procedure.
- **Beginning, ending, and issuing requests.** A report request begins with the TABLE FILE command and ends with the END command. The commands and phrases between the beginning and end of a request define the contents and format of a report.
- **Report syntax and usage.** You can use FOCUS reporting to select data for your report and manipulate the way the report displays the data.
- **Specifying fields and columns.** Each column in your report represents a data field. FOCUS is flexible, providing you with several ways to refer to and specify fields.

Before You Begin: Requirements

There are only two simple requirements for reporting:

- **Data.** You need data from which to report. If the data is protected by an underlying security system, you may need permission to report from the data source.

  In addition, the FOCUS session or server must be able to locate the data source.

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

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

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

  ```
  CHECK FILE filename PICTURE RETRIEVE
  ```

  In the picture, segments are shown in the order in which FOCUS retrieves them to generate a report. Four fields of each segment are displayed. The CHECK FILE command is described in more detail in the *Describing Data* manual.

Developing Your Report Request

You can use an editor to create your TABLE request and save it as a stored procedure. This enables you to edit your request at any time.

Stored procedures are described in more detail in the *Developing Applications* manual.

Beginning, Ending, and Issuing a Report Request

The FOCUS reporting language is very flexible, requiring only a command to mark the beginning of the request and identify the data source

```
TABLE FILE filename
```

and a command to mark the end of the request:

```
END or RUN
```

All of the other parts of the request are optional: you only need to include the commands and phrases which produce the report functions you want. For example, you need only include a sorting phrase if you want your report to be sorted.

The following sections introduce you to these optional functions. All of the functions are described in detail in the remainder of this chapter.

Beginning a Report Request

To begin a report request, use the TABLE command.

The syntax is

```
TABLE FILE filename
```

where:

filename
Specifies a data file for the report.

You can report from many different types of data files, including the following:

- Relational databases, such as DB2, Teradata, and Oracle.
- Hierarchical databases, such as IMS.
- Network databases, such as CA-IDMS/DB.
- Indexed data files, such as VSAM.
- Sequential data files, both fixed-format and comma-delimited.
- Native FOCUS databases.
- Extract files produced by previous report requests.
- Structures of joined data files.

Ending a Report Request

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

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

If you plan to issue consecutive report requests against the same data file during one FOCUS session, you have the option of using the RUN command. RUN keeps the TABLE facility and the data source active for the duration of the TABLE session. After viewing one report you do not need to repeat the TABLE command to produce another report. You terminate the TABLE session by issuing the END command after the last request.

You may issue all the commands and phrases between the TABLE FILE and END commands on a single line or on multiple lines. Except for the END command, the location of line-breaks between words is entirely arbitrary.

The Parts of a Report Request

The FOCUS reporting language is very flexible, requiring only a command to mark the beginning of the request and identify the data source,

```
TABLE FILE filename
```

and a command to mark the end of the request:

```
END
```

All of the other parts of the request are optional; you only need to include the commands and phrases which produce the report functions you want. For example, you need only include a sorting phrase if you want your report to be sorted. The options include:

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

- **Storing and reusing the results.** You can store your report as a data source on which you can make additional queries. This is especially helpful for creating a subset of your data source and for generating two-step reports. You can also format the new data source for use by other data processing tools such as spreadsheets and word processors.

Displaying Data

Reporting, at the simplest level, retrieves field values from a data source and displays these values. FOCUS provides you with three ways to do this:

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

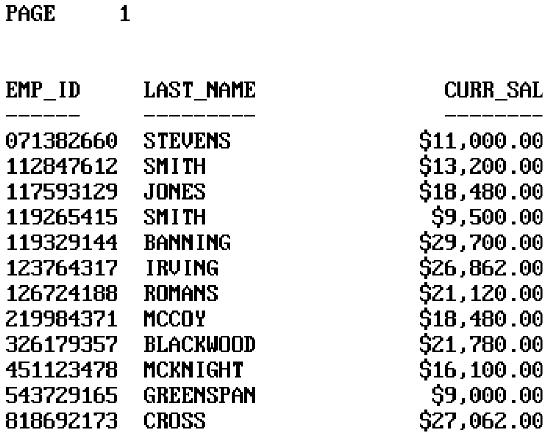
Example Displaying Data

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

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

Notice that each field named in the PRINT command has its own report column.

The request generates the following from a FOCUS application.



```
PAGE     1


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

Sorting Report Columns

Sorting a report enables you to organize a column’s information in a chosen order. FOCUS displays the sort field—the field that controls the sorting order—at the left of the report. Sort fields are displayed when their values change.

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

Example Sorting Report Columns

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

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

This request generates the following report:

```
PAGE     1


LAST_NAME        EMP_ID             CURR_SAL
---------        ------             --------

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

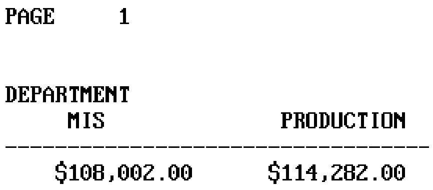
FOCUS displays the sort field—the field that controls the sorting order—at the left of the report. Note that two rows of the report are assigned to the sort value SMITH because there are two employees named Smith. Sort fields are displayed when their values change.

Example Sorting Report Rows

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

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

The request generates this report.



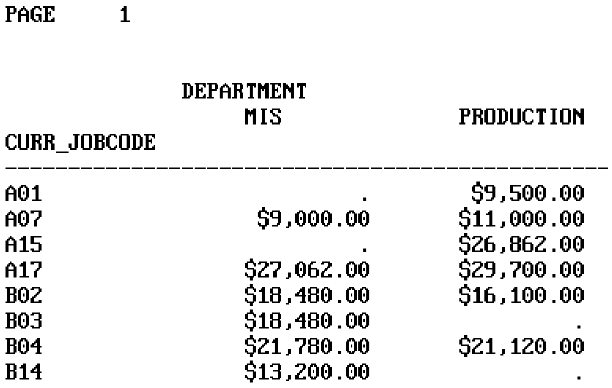
```
PAGE     1


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

Example Sorting by Rows and Columns

If you want to show each employee’s current salary and job code, and you want to organize the report by department, you could issue the following request:

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

This request generates the following report.



```
PAGE     1


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

Selecting Records

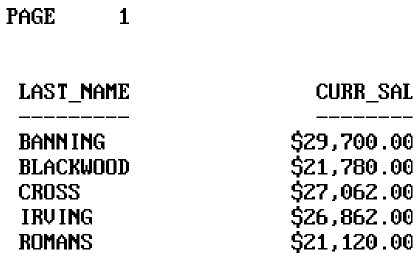
When you generate a report, you may not want to include every record. Selecting records enables you to define a subset of the data source based on your criteria and then report on that subset. Your selection criteria can be as simple or complex as you wish.

Example **Selecting Records**

If you want to report on employee salaries, you may be interested only in employees earning more than $20,000 a year. You can select the records for these employees using the WHERE phrase:

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

Only employees earning more than $20,000 appear in the report:



```
PAGE     1


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

Showing Subtotals and Totals

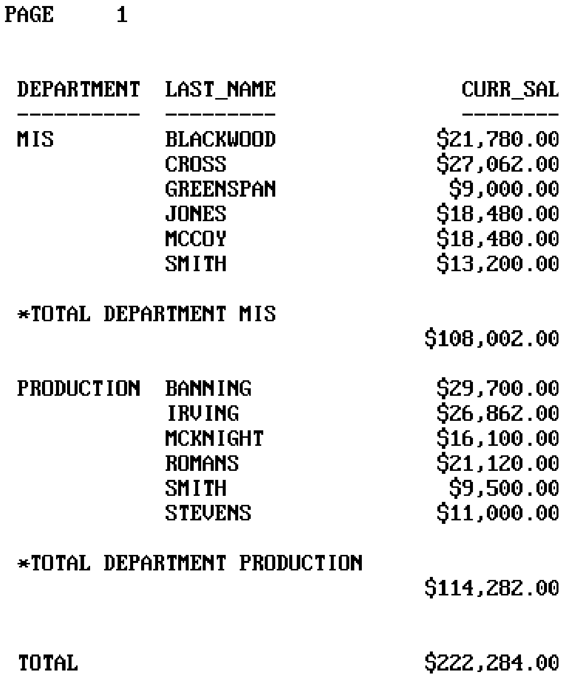
You can include subtotals and totals for some of the fields in your report.

Example **Showing Subtotals and Totals**

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

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

The request generates this report:



```
PAGE     1


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

 *TOTAL DEPARTMENT MIS
                                 $108,002.00

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

 *TOTAL DEPARTMENT PRODUCTION
                                 $114,282.00


 TOTAL                           $222,284.00
```

Creating Temporary Fields

When you create a report, you are not limited to the fields that already exist in the data source. You can create new fields that are based on the values of existing fields.

Once the temporary field is defined, you can use it in subsequent report requests in the same FOCUS session until it is cleared.

You can also define temporary fields in a Master File using the DEFINE attribute, described in the *Describing Data* manual, and in a report request using the COMPUTE command, described in Chapter 6, *Creating Temporary Fields.*

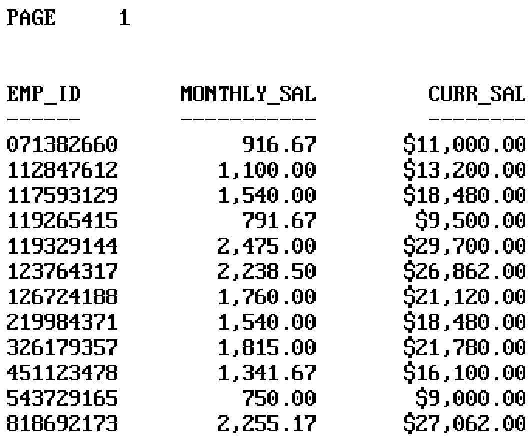
Example Creating Temporary Fields

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

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

Once the MONTHLY_SAL field is defined, you can use it in subsequent report requests in the same FOCUS session until it is cleared.

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



```
PAGE     1


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

Customizing the Presentation

There are two aspects of a successful report: the information presented and how it is presented. A report that identifies related groups of information and draws attention to important facts will be more effective than one that simply shows columns of data. FOCUS offers many formatting options, including features that enable you to do the following:

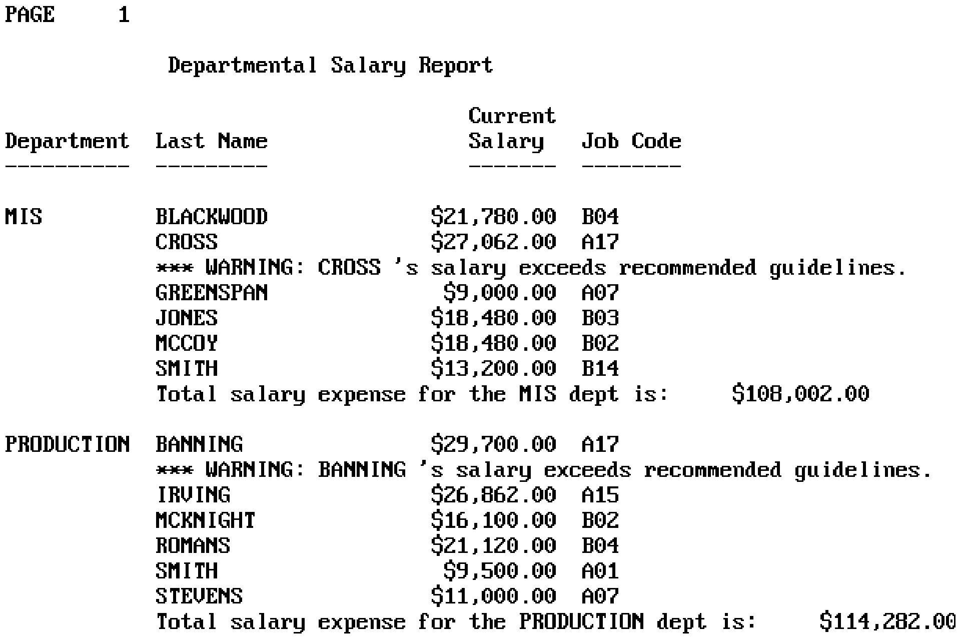
- Highlight a group of related information and separate it from other groups by inserting blank lines or dashes between each group. You can accomplish this using the SKIP-LINE and UNDER-LINE phrases.
- Give column titles more meaningful names by using the AS phrase.
- Create headers and footers for different levels of the report—including each sort group, each page, and the entire report—by using the HEADING, SUBHEAD, FOOTING, and SUBFOOT phrases.
- Dynamically control the display of subtotals, headers, and footers based on the data appearing in the report by using the WHEN phrase.

Example Customizing the Presentation

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

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

The report is shown below:



```
PAGE     1

             Departmental Salary Report

                                     Current
Department  Last Name                Salary   Job Code
----------  ---------                -------  --------

MIS         BLACKWOOD             $21,780.00  B04
            CROSS                 $27,062.00  A17
            *** WARNING: CROSS ’s salary exceeds recommended guidelines.
            GREENSPAN              $9,000.00  A07
            JONES                 $18,480.00  B03
            MCCOY                 $18,480.00  B02
            SMITH                 $13,200.00  B14
            Total salary expense for the MIS dept is:     $108,002.00

PRODUCTION  BANNING               $29,700.00  A17
            *** WARNING: BANNING ’s salary exceeds recommended guidelines.
            IRVING                $26,862.00  A15
            MCKNIGHT              $16,100.00  B02
            ROMANS                $21,120.00  B04
            SMITH                  $9,500.00  A01
            STEVENS               $11,000.00  A07
            Total salary expense for the PRODUCTION dept is:     $114,282.00
```

Output

Once you generate a report, you still need to display it. FOCUS offers facilities for displaying your report:

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

Of course, you can easily direct the same report to both the screen and the printer by switching display modes and using the RETYPE command. Or you can choose not to display your report at all, and instead store the results as a data file using the HOLD, PCHOLD, SAVE, or SAVB command.

Storing and Reusing the Results

By using the HOLD, PCHOLD, SAVE, and SAVB commands, you can store your report as a data source on which you can make additional queries. This capability is especially helpful for creating a subset of your data and for generating two-step reports. You can also format the new data source for use by other data processing tools such as spreadsheets and word processors.

Referring to Fields

You refer to fields in several parts of a report request, including the following:

- The display command—specifying which fields you want to display.
- The sort phrase—specifying which fields you want to use to sort the report.
- The selection phrase—declaring which fields you want to use to select records.

FOCUS offers several ways to refer to fields in a report request, and offers help for remembering what the names of all the fields are.

Referring to a Single Field

FOCUS gives you the flexibility of referring to a field in any one of the following ways:

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

Example Referring to a Single Field

Consider the following report requests:

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT
END
```

```
TABLE FILE EMPLOYEE
PRINT DPT
END
```

```
TABLE FILE EMPLOYEE
PRINT DEP
END
```

DEPARTMENT is the complete field name. DPT is the alias, and DEP is a truncation of DEPARTMENT. All these examples produce the same results.

Example Incorrectly Using a Truncation

When a truncation is used, it must be unique; if it is not unique, an error message is displayed. For example:

```
TABLE FILE EMPLOYEE
PRINT D
END
```

Because other field names begin with “D” in the EMPLOYEE Master File, “D” is not unique, and the following message appears:

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

Referring to Long and Qualified Fields

Field names and aliases may have a maximum length of 66 characters, including up to two qualifiers and qualification characters. Defined and computed field names may also be up to 66 characters in length. Text fields and indexed field names in Master Files are limited to 12 characters. However, the aliases for text and indexed fields may be up to 66 characters long. Field names up to 66 characters long are displayed as column titles in TABLE reports, unless a TITLE attribute or an AS phrase is used.

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

```
EMPLOYEE.EMPINFO.EMP_ID
```

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

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

```
SET FIELDNAME = type
```

where:

type

Can be one of the following:

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

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

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

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

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

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

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

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

Referring to All of the Fields in a Segment: SEG.

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

Example Referring to a Segment

For example, the segment PRODUCT described in a Master File contains the PROD_CODE, PROD_NAME, PACKAGE, and UNIT_COST fields.

```
SEGMENT=PRODUCT
FIELDNAME = PROD_CODE
FIELDNAME = PROD_NAME
FIELDNAME = PACKAGE
FIELDNAME = UNIT_COST
```

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

```
TABLE FILE PROD
PRINT PROD_CODE AND PROD_NAME AND PACKAGE AND UNIT_COST
END
```

```
TABLE FILE PROD
PRINT SEG.PROD_CODE
END
```

```
TABLE FILE PROD
PRINT SEG PROD_CODE
END
```

```
TABLE FILE PROD
PRINT SEGMENT PROD_CODE
END
```

Displaying a List of Field Names: ?F

If you want to see a list of all the fields that are included in the currently active data source, you can issue the field name query from within a command:

```
?F
```

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

Listing Field Names, Aliases, and Format Information: ?FF

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

- From the FOCUS command level.
- When entering a TABLE, MODIFY FILE, or GRAPH request online.
- When you are in the FSCAN environment.

Note:

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

Specifying Columns

Each column in a report represents a real or temporary field that you specified in the report request. You can control a column's title and format and indirectly control its width.

Specifying the Title of a Report Column

The title at the top of a report column defaults to the name, as defined in the Master File, of the field that is displayed in the column. You can specify a different title in either of the following ways:

- Use the TITLE attribute in the Master File. Each time the field is displayed in a report request, the title specified in the Master File appears at the top of the column.

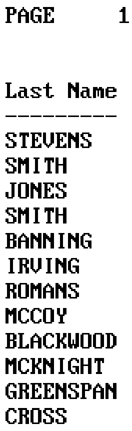
  You can still override the TITLE attribute in an individual report request by using the AS phrase.
- Use the AS phrase in the report request. The AS phrase enables you to specify any title for any report column, overriding the field name default and the TITLE attribute. The AS phrase is described in more detail in Chapter 10, *Customizing Tabular Reports*.

Example Specifying the Title With the AS Phrase

The following report request uses the AS phrase to rename the LAST_NAME column:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AS 'Last Name'
END
```

The request renames the LAST_NAME column as Last Name.



```
PAGE     1


Last Name
---------
STEVENS
SMITH
JONES
SMITH
BANNING
IRVING
ROMANS
MCCOY
BLACKWOOD
MCKNIGHT
GREENSPAN
CROSS
```

Changing the Format of a Report Column

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

Syntax How to Change Column Format

The syntax is

`fieldname` `[`*`alignment`*`] [`*`/format`*`]`

where:

`fieldname`

Is a display field—that is, a field displayed by the PRINT, LIST, SUM, or COUNT command, a row-total, or a column-total.

`alignment`

Specifies the position of the column title.

`/R` specifies a right justified column title.

`/L` specifies a left justified column title.

`/C` specifies a centered column title.

`format`

Is any valid field format, preceded by a slash (/). Field formats are described in the *Describing Data* manual. Field formats cannot be used with a column total.

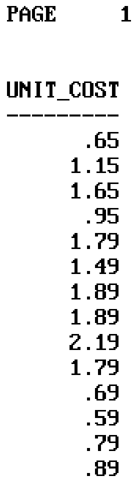
Reference Usage Notes for Changing Column Format

- Each time you reformat a column, the field is counted twice against the limit of 256 display fields in a single report.
- If you create an extract file from the report—that is, a HOLD, PCHOLD, SAVE, or SAVB file—the extract file will contain fields for both the original format and the redefined format. Extract files are described in Chapter 12, *Saving and Reusing Your Report Output.*
- Format redefinition may not be used on a field in a BY or ACROSS phrase or with SUM CNT.field name.
- When the size of a word in a text field instance is greater than the format of the text field in the Master File, the word wraps to a second line, and the next word begins on the same line.
- Format redefinition cannot be used with unique truncation.
- You may specify justification for display fields, BY fields, and ACROSS fields. For ACROSS fields, data values, not column titles, are justified as specified.
- For display commands only, the justification parameter may be combined with a format specification. The format specification may precede or follow the justification parameter.
- If a title is specified with an AS phrase or in the Master File, that title will be justified as specified in FORMAT.
- When multiple ACROSS fields are requested, justification is performed on the lowest ACROSS level only. All other justification parameters for ACROSS fields are ignored.

Example Changing a Column's Format

For example, the UNIT_COST field has a format of D5.2M as defined in the PROD Master File. To prevent the display of the floating dollar sign, the field format can be redefined as follows:

```
TABLE FILE PROD
PRINT UNIT_COST/D5.2
END
```

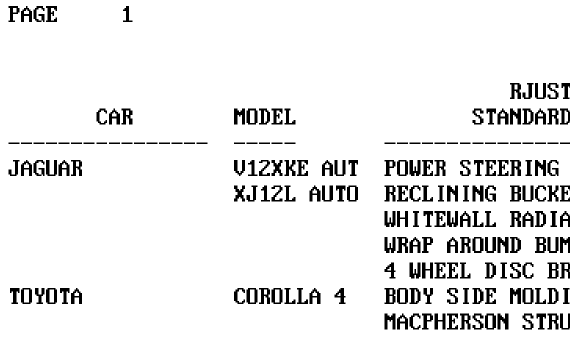


```
PAGE     1


UNIT_COST
---------
      .65
     1.15
     1.65
      .95
     1.79
     1.49
     1.89
     1.89
     2.19
     1.79
      .69
      .59
      .79
      .89
```

Example Using Multiple Format Specifications

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

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



```
PAGE     1


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

Determining the Width of a Report Column

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

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

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

Note: The default spacing can be overridden by using IN or OVER. You can also use SET SPACES to control column spacing, for more information see Chapter 10, *Customizing Tabular Reports*.

Issuing Report Requests

You can issue report requests in a stored procedure of FOCUS commands—also known as a FOCEXEC—or interactively at the FOCUS command line.

You can create a FOCUS stored procedure using TED or any other editor; TED is described in the *Overview and Operating Environments* manual. Stored procedures are discussed in more detail in the *Developing Applications* manual.

Once a report request is saved in a stored procedure, you can issue the request by executing the procedure using the EXEC or EX command. For example, if you stored the report request in a file called TEST, you would execute the stored procedure with the following command at the FOCUS command line:

```
EX TEST
```

When you plan to generate multiple reports from a single request, storing the request in a FOCUS stored procedure will save you typing and development time.

If you issue report requests interactively at the FOCUS command prompt, rather than from a procedure, FOCUS will provide online error correction with help text.

Online error correction is available after you enter the keyword TABLE correctly. For example, if you enter

```
TABLE FI EMPLOYEE
```

at the command prompt, FOCUS will respond with the error message:

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

Enter your correction at the REPLY prompt and FOCUS will wait for your next command. For this example, the correct reply is:

```
FILE
```

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

```
HELP
```

or

```
?
```

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

FOCUS will scan and process every command you enter, and generate a report immediately after you enter the END or RUN command.

When the value of the SET command’s MESSAGE parameter is ON (the default value, as described in the *Developing Applications* manual), FOCUS displays—at the beginning of each report—the number of records retrieved from the data source and the number of lines displayed in the report.

2 Displaying Report Data

Topics:

- Display Commands: General Syntax
- Printing Individual Values: PRINT and LIST
- Adding Values: SUM, WRITE, and ADD
- Counting Values: COUNT
- Expanding Byte Precision for COUNT and LIST
- Manipulating Display Fields: Prefix Operators
- Manipulating Display Fields: WITHIN

Reporting, at the simplest level, retrieves field values from a data source and displays those values. FOCUS provides you with three ways to do this:

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

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

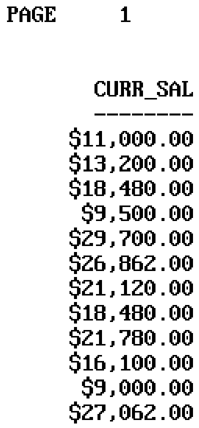
Note: The AUTOPATH setting may affect the order of displayed data.

Example Printing Field Values

For example, assume that you want to report on salary information. You can use the PRINT command to print each individual salary.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
END
```

The request produces this report.



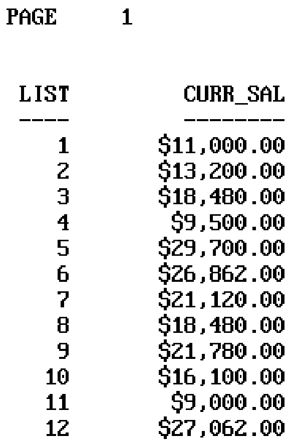
```
PAGE     1


       CURR_SAL
       --------
     $11,000.00
     $13,200.00
     $18,480.00
      $9,500.00
     $29,700.00
     $26,862.00
     $21,120.00
     $18,480.00
     $21,780.00
     $16,100.00
      $9,000.00
     $27,062.00
```

Example Listing Field Values

You can also show the order of each salary as it appears in the report column by using the LIST command.

```
TABLE FILE EMPLOYEE
LIST CURR_SAL
END
```

The request produces this report.



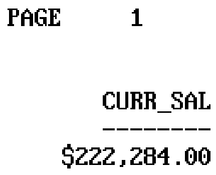
```
PAGE     1


 LIST         CURR_SAL
 ----         --------
    1       $11,000.00
    2       $13,200.00
    3       $18,480.00
    4        $9,500.00
    5       $29,700.00
    6       $26,862.00
    7       $21,120.00
    8       $18,480.00
    9       $21,780.00
   10       $16,100.00
   11        $9,000.00
   12       $27,062.00
```

Example Summing Field Values

You can display the total of all salaries by using the SUM command.

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

The request produces this report.



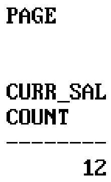
```
PAGE     1


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

Example Counting Field Values

You can even determine how many salaries have been entered into the EMPLOYEE database by using the COUNT command.

```
TABLE FILE EMPLOYEE
COUNT CURR_SAL
END
```

The request produces this report.



```
PAGE     1


CURR_SAL
COUNT
--------
      12
```

Display Commands: General Syntax

The syntax of a display command is

```
display  {[THE] fieldname1 [AND] [THE] fieldname2 ...|*}
```

where:

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

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

  There can be a maximum of 256 verb objects in a request. This number includes all named fields, whether printed or not, such as data source fields, DEFINEd fields, COMPUTEd fields, certain internal fields, like TABPAGENO, and fields used in headings and footings. The fields appear in the report in the same order in which they are specified in the report request. For example, the report column for field1 will appear first, followed by the report column for field2.

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

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

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

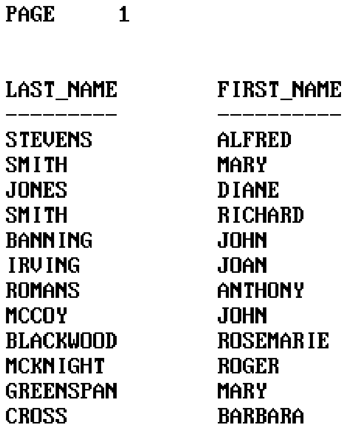
*
: Applies the display command to every field in the left path of the database.

Example Displaying Multiple Fields

For example, by specifying two display fields, you can print the full names of all employees.

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

The request produces this report.



```
PAGE     1


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

Printing Individual Values: PRINT and LIST

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

You can easily display all of the fields in the data source by specifying an asterisk (*) wildcard instead of a specific field name, as described in *Printing All Fields: The * Wildcard* on page 2-6.

For all PRINT and LIST requests, the number of records retrieved and the number of lines displayed are the same. In addition, there is no order to the report rows. The PRINT and LIST commands simply display all the values of the selected fields found in the database.

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

The values are displayed in the order in which they are accessed from the database.

Alternatively, you can sort the values using the BY or ACROSS sort phrases, described in Chapter 4, *Sorting Tabular Reports.*

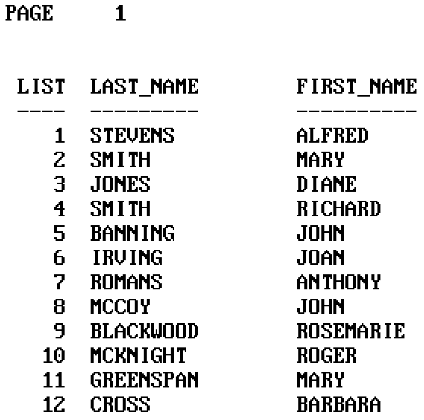
Note: When LIST is used in a request that includes a sort phrase, the list counter is reset to 1 every time the value in the outermost sort field changes. See Chapter 4, *Sorting Tabular Reports*, for more information on sorting.

Example Listing Records

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

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

This report request produces the following report.



```
PAGE     1


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

Printing All Fields: The * Wildcard

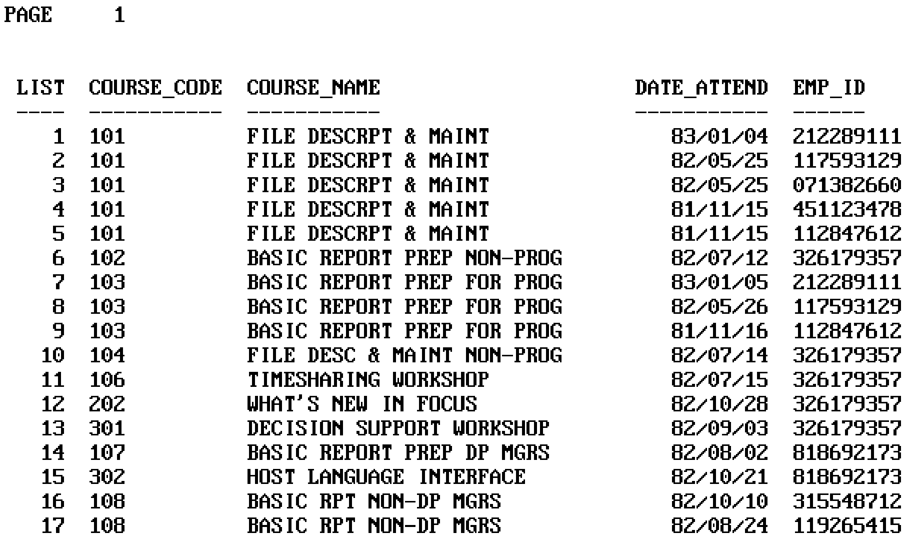
You can easily display all of the fields in the left path of the data source by specifying an asterisk (*) wildcard instead of a specific field name. For additional information about Master File structures and segment paths, including left paths and short paths, see the *Describing Data* manual.

Example Printing All Fields

For example, the following request produces a report displaying all of the fields of EDUCFILE.

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

The request generates this report:



```
PAGE     1


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

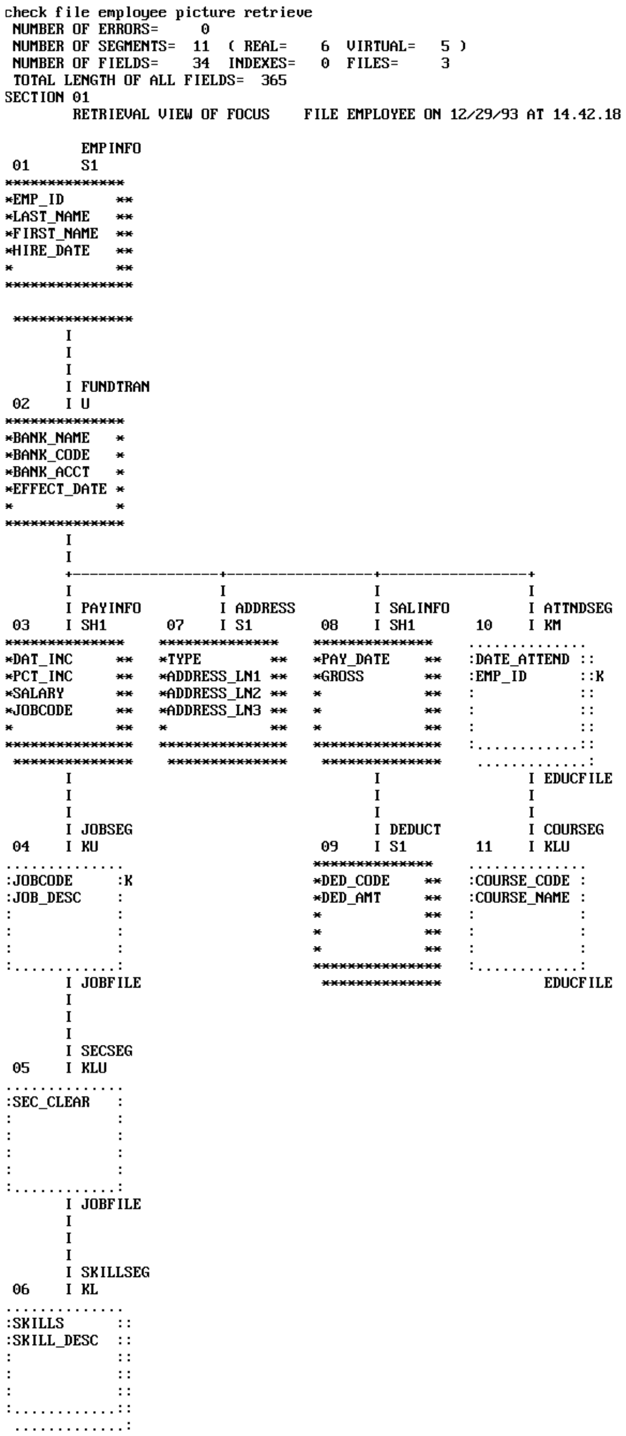
Example Using the CHECK Command

To see the structure of the EMPLOYEE database, which is joined to the JOBFILE and EDUCFILE databases, issue the following command:

```
CHECK FILE EMPLOYEE PICTURE RETRIEVE
```

The command generates the following picture. Note that a unique segment such as FUNDTRAN is treated as a descendant segment.

Note that FUNDTRAN and SECSEG are unique segments and are, therefore, treated as part of their parents.



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

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

 **************
       I
       I
       I
       I FUNDTRAN
 02    I U
**************
*BANK_NAME   *
*BANK_CODE   *
*BANK_ACCT   *
*EFFECT_DATE *
*            *
**************
       I
       I
       +-----------------+-----------------+-----------------+
       I                 I                 I                 I
       I PAYINFO         I ADDRESS         I SALINFO         I ATTNDSEG
 03    I SH1       07    I S1        08    I SH1       10    I KM
**************    **************    **************    ..............
*DAT_INC     **   *TYPE        **   *PAY_DATE    **   :DATE_ATTEND ::
*PCT_INC     **   *ADDRESS_LN1 **   *GROSS       **   :EMP_ID      ::K
*SALARY      **   *ADDRESS_LN2 **   *            **   :            ::
*JOBCODE     **   *ADDRESS_LN3 **   *            **   :            ::
*            **   *            **   *            **   :            ::
***************   ***************   ***************   :............::
 **************    **************    **************    .............:
       I                                   I                 I EDUCFILE
       I                                   I                 I
       I                                   I                 I
       I JOBSEG                            I DEDUCT          I COURSEG
 04    I KU                         09    I S1        11    I KLU
..............                      **************    ..............
:JOBCODE     :K                     *DED_CODE    **   :COURSE_CODE :
:JOB_DESC    :                      *DED_AMT     **   :COURSE_NAME :
:            :                      *            **   :            :
:            :                      *            **   :            :
:            :                      *            **   :            :
:............:                      ***************   :............:
       I JOBFILE                     **************         EDUCFILE
       I
       I
       I
       I SECSEG
 05    I KLU
..............
:SEC_CLEAR   :
:            :
:            :
:            :
:            :
:............:
       I JOBFILE
       I
       I
       I
       I SKILLSEG
 06    I KL
..............
:SKILLS      ::
:SKILL_DESC  ::
:            ::
:            ::
:            ::
:............::
 .............:
```

Example Printing Fields From Multi-Path Data Sources

For example, when you issue the following table request

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

you will receive this message:

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

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

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

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

Each field in this list appears in segments on the left path of the EMPLOYEE database.

Adding Values: SUM, WRITE, and ADD

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

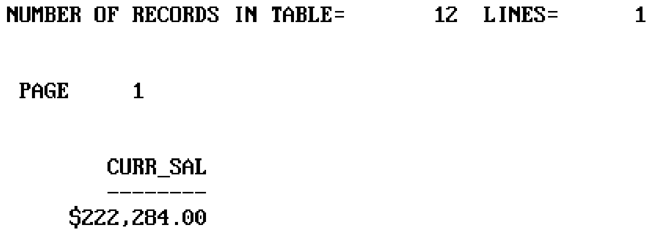
When you use sum, multiple records are read from the data source, but only one summary line is produced. If you use SUM with a non-numeric field—such as an alphanumeric, text, or date field—SUM will not add the values; instead, it will display the last value retrieved from the data source. This may not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

Example Adding Values

For example, this request adds all the values of the field CURR_SAL.

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

The request generates this report:



```
NUMBER OF RECORDS IN TABLE=       12  LINES=      1


 PAGE     1


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

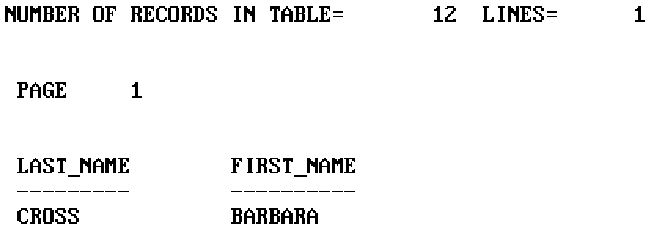
Notice that the number of lines in the report is less than the number of records from the data file. It took a total of 12 records to get the results in the report, but only one summary line is printed.

Example Adding Non-Numeric Values

For example, this request attempts to add non-numeric fields:

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

Any request for aggregation on non-numeric data returns the last record from the data source. Any request for aggregation on a smart date field also returns the last record from the data source.



```
NUMBER OF RECORDS IN TABLE=       12  LINES=      1


 PAGE     1


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

Counting Values: COUNT

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

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

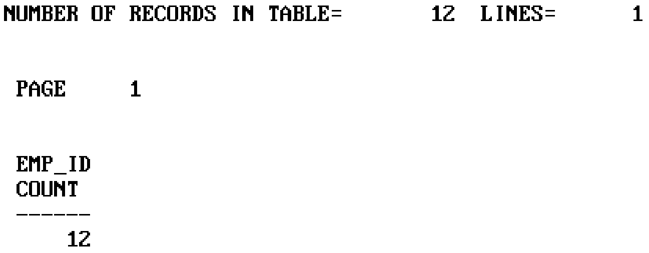
Note: By default, a COUNT field is a five-digit integer. You can reformat it using COMPUTE.

Example Counting Values

To determine how many employees are in the EMPLOYEE database, you could count the instances of EMP_ID, which is the employee identification number.

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

This request produces this report, which states that there are 12 EMP_IDs in the database.



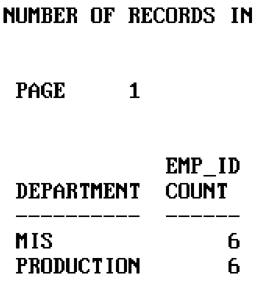
```
NUMBER OF RECORDS IN TABLE=       12  LINES=      1


 PAGE     1


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

Example Counting Values With a Sort Phrase

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

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

The request produces this report. This report tells us that of the 12 EMP_IDs in the database, six are from the MIS department and six from Production.



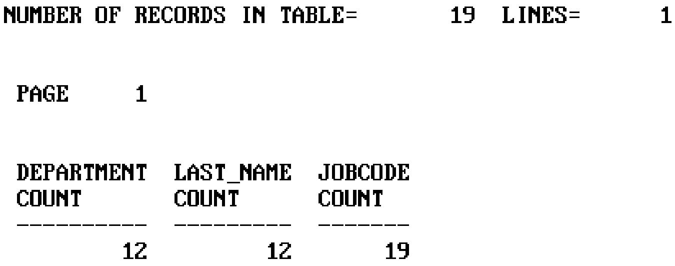
```
NUMBER OF RECORDS IN TABLE=       12  LINES=      1


 PAGE     1


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

Example Counting Instances of Data

Consider the following example:

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

This request produces the following report:



```
NUMBER OF RECORDS IN TABLE=       19  LINES=      1


 PAGE     1


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

The EMPLOYEE database contains data on 12 employees, with one root segment instance for each employee. While there are two departments, the DEPARTMENT field is in the root segment, so there are 12 instances of the DEPARTMENT field. Similarly, there are 19 instances of the JOBCODE field because this code, which is stored in a descendant segment, is entered each time an employee receives a salary increase.

Counting Segment Instances: The * Wildcard

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

COUNT * accomplishes this by counting the values of the first field in the segment. Instances with a null value in the first field are not counted (when SET MISSING=ON). Null values (sometimes called missing data) are discussed in Chapter 13, *Handling Records With Missing Field Values.*

Segment instances in short paths are not counted by COUNT *, regardless of the value of the ALL parameter of the SET command. The SET ALL command and short paths are described in more detail in Chapter 13, *Handling Records With Missing Field Values*.

Example Counting Segments From a Multi-Path Database

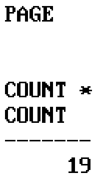
If you use COUNT * with a multi-path data source (such as EMPLOYEE in this example),

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

FOCUS returns the following message:

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

The request produces this report. COUNT * counted the number of instances of the SKILLSEG segment, which is the lowest segment in the left path of the EMPLOYEE database structure (that is, the EMPLOYEE database joined to the JOBFILE and EDUCFILE databases). You can see a picture of the path structure in *Printing Individual Values: PRINT and LIST* on page 2-5.



```
PAGE     1


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

Expanding Byte Precision for COUNT and LIST

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

Previously, if the number of records retrieved for a field exceeded 5 bytes, asterisks were displayed in the report. This indicated 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 SET COUNTWIDTH = OFF:

```
TABLE FILE filename
COUNT Fldxx
BY Fldyy
END
```

```
       FLDxx

Fldyy  COUNT

value  *****
```

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

```
TABLE FILE filename
COUNT Fldxx
BY Fldyy
END
```

```
           FLDxx

Fldyy      COUNT

value  999999999
```

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

Manipulating Display Fields: Prefix Operators

FOCUS provides a variety of ways to perform calculations. You can use prefix operators to perform calculations directly on the values of fields. You can:

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

Syntax How to Use Prefix Operators

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

```
{SUM|COUNT} prefix.fieldname
```

where:

prefix
Is any prefix operator.

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

Reference Usage Notes for Prefix Operators

- Because PRINT and LIST display individual field values, not an aggregate value, they are not used with prefix operators.
- To sort by the results of a prefix command, store the results in a HOLD file and then issue a new report request against that HOLD file.
- The WITHIN phrase is very useful when using prefixes.
- You can use the results of prefix operators in COMPUTE commands.
- With the exception of CNT. and PCT.CNT., resulting values have the same format as the field against which the prefix operation was performed.
- Text fields can only be used with the FST. and LST. prefix operators.

Averaging Values of a Field: AVE.

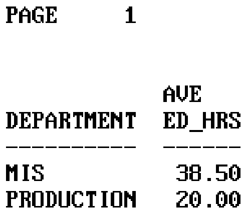
The AVE. prefix computes the average value of a particular field. The computation is performed at the lowest sort level of the display command. It is computed as the sum of the field values within a sort group divided by the number of records in that sort group. If the request does not include a sort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

Example Averaging Values of a Field

For example, this request calculates the average number of education hours spent in each department.

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

This request produces the following report:



```
PAGE     1


            AVE
DEPARTMENT  ED_HRS
----------  ------
MIS          38.50
PRODUCTION   20.00
```

Averaging the Sum of Squared Fields: ASQ.

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

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

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

If the field format is packed, ASQ. is not supported and becomes AVE.

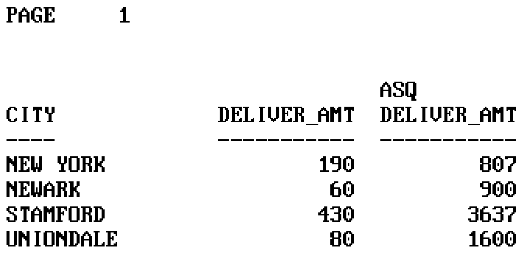
If the field format is integer and you get a large set of numbers, the ASQ. result may be negative as a result of field overflow.

Example Averaging the Sum of Squared Fields

For example, this request calculates the sum and the sum of squared fields for the DELIVER_AMT field.

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

This request creates the following report:



```
PAGE     1


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

Calculating the Maximum and Minimum of Field Values: MAX. and MIN.

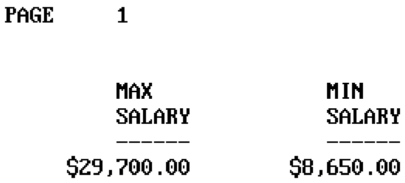
The prefixes MAX. and MIN. produce the maximum and minimum values, respectively, within a sort group. If the request does not include a sort phrase, MAX. and MIN. produce the maximum and minimum values for the entire report.

Example Calculating Maximum and Minimum Field Values

For example, this report request calculates the maximum and minimum values of SALARY.

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

The request produces the following report:



```
PAGE     1


         MAX              MIN
         SALARY           SALARY
         ------           ------
     $29,700.00       $8,650.00
```

Calculating Column and Row Percents: PCT. and RPCT.

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

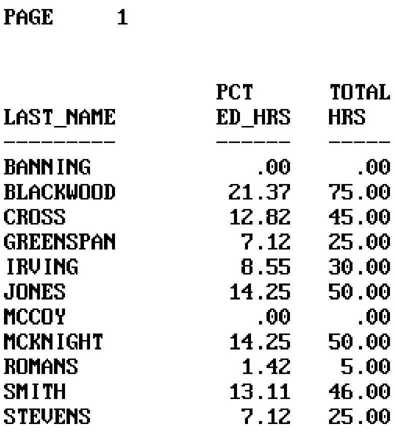
You can also determine percentages for row values; for each individual value in a row that has been sorted using the ACROSS phrase, the RPCT. operator calculates what percentage it makes up of the row’s total value. The percentage values have the same format as the original field.

Example Calculating Column Percents

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

```
TABLE FILE EMPLOYEE
SUM PCT.ED_HRS ED_HRS AS 'TOTAL,HRS' BY LAST_NAME
END
```

This request generates the following report:



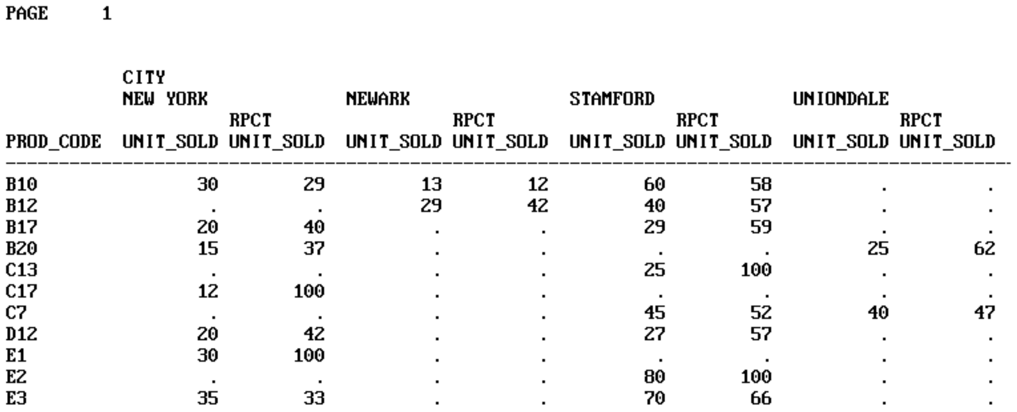
```
PAGE     1


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

Example Calculating Row Percents

The request is as follows:

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

Part of the report that results is displayed here:



```
PAGE     1


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

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

Producing a Direct Percent of a Count: PCT.CNT.

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

```
PCT.CNT.fieldname
```

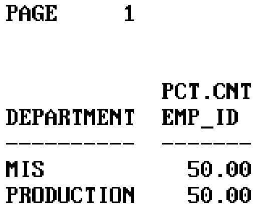
The format is a decimal value of six digits with two decimal places (F6.2).

Example Producing a Direct Percent of a Count

The request is as follows:

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

This request produces the following report:



```
PAGE     1


            PCT.CNT
DEPARTMENT  EMP_ID
----------  -------
MIS           50.00
PRODUCTION    50.00
```

Aggregating and Listing Unique Values: DST.

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

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

Syntax How to Use the Distinct Operator

The syntax is:

```
{command} DST.fieldname
```

or

```
SUM [operator].DST.fieldname
```

where:

command
Is SUM, PRINT, or COUNT.

DST.
Indicates the distinct operator.

fieldname
Indicates the verb object or field name.

operator
Indicates SUM., CNT., or AVE.

Example Using the Distinct Operator

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

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

or

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

The output is:

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

       9
```

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

When used with PRINT, DST. acts in the same manner as a BY statement. It can be attached to several PRINT verb objects, but not more than 32 (the current limit for sort fields in FOCUS). DST. verb objects must precede all non-distinct verb objects named by the PRINT verb. If not, you will get the following error:

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

Reference Restrictions

- The following error occurs if you use the prefix operators CNT., SUM., and AVE. with any other display command:

  ```
  (FOC1853) CNT/SUM/AVE.DST CAN ONLY BE USED WITH AGGREGATION VERBS
  ```

- The following error occurs if you use DST. in a MATCH or TABLEF command:

  ```
  (FOC1854) THE DST OPERATOR IS ONLY SUPPORTED IN TABLE REQUESTS
  ```

- The following error occurs if you code more than one DST. verb object operator for the SUM verb:

  ```
  (FOC1856) ONLY ONE DISTINCT FIELD IS ALLOWED IN AGGREGATION
  ```

- The following error occurs if you reformat a BY field (when used with the PRINT verb, the DST.*fieldname* becomes a BY field):

  ```
  (FOC1862) REFORMAT DST.FIELD IS NOT SUPPORTED WITH PRINT
  ```

- The following error occurs if you use the DST. verb object operator in an ACROSS or FOR statement:

  ```
  (FOC1864) THE DST OPERATOR IS NOT SUPPORTED FOR ACROSS OR FOR
  ```

- The following error occurs if you use a multi-verb request, SUM DST.*fieldname* BY *field* PRINT *fld* BY *fld* (a verb object operator used with the SUM verb must be at the lowest level of aggregation):

  ```
  (FOC1867) DST OPERATOR MUST BE AT THE LOWEST LEVEL OF AGGREGATION
  ```

- The DST. operator may not be used as part of a HEADING or a FOOTING.
- TABLE requests that contain the DST. operator are not converted to TABLEF.

Retrieving First and Last Records: FST. and LST.

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

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

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

For more information on segment types and file design, see the *Describing Data* manual. If you wish to reorganize the data in the data source or restructure the data source while reporting, see Chapter 16, *Improving Data Retrieval*.

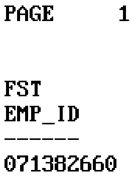
Example

Retrieving the First Record

The request is:

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

This request produces the following report.



```
PAGE     1


FST
EMP_ID
------
071382660
```

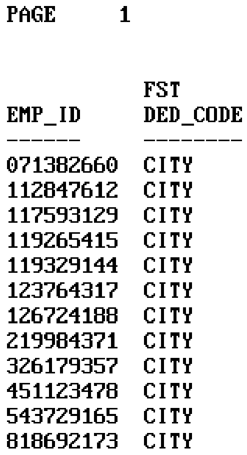
Example

Segment Types and Retrieving Records

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

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

The request generates the following report. However, the phrase SUM LST.DED_CODE would have retrieved the last logical record for DED_CODE for each employee.



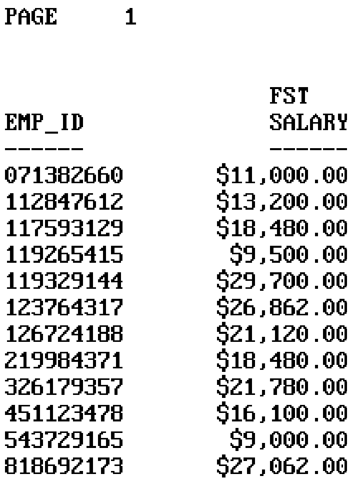
```
PAGE     1


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

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

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

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

This request produces the following report. However, the phrase SUM LST.SALARY would have retrieved the last logical record for SALARY for each employee.



```
PAGE     1


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

Summing and Counting Values: CNT., SUM., TOT.

You can count occurrences and summarize values all in one display command using the prefixes CNT., SUM., and TOT. Just like the COUNT command, CNT. counts the occurrences of the one field it prefixes, just like the SUM command, SUM. sums the values of a particular field.

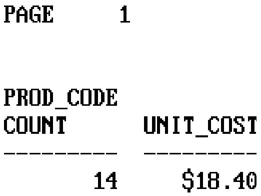
The prefix operator TOT. is best used when you want to put data into headings. For more information, see Chapter 10, *Customizing Tabular Reports*.

Example Counting Values With CNT

For example, the following request,

```
TABLE FILE PROD
SUM CNT.PROD_CODE AND UNIT_COST
END
```

creates this report.



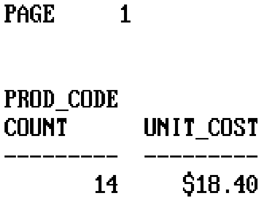
```
PAGE     1


PROD_CODE
COUNT      UNIT_COST
---------  ---------
       14     $18.40
```

Example Summing Values With SUM

For example, the following request,

```
TABLE FILE PROD
COUNT PROD_CODE AND SUM.UNIT_COST
END
```

creates this report.



```
PAGE     1


PROD_CODE
COUNT      UNIT_COST
---------  ---------
       14     $18.40
```

Manipulating Display Fields: WITHIN

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

Syntax How to Use WITHIN to Manipulate Display Fields

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

The syntax is

```
{SUM|COUNT} display_field WITHIN sort_field BY sort_field
```

where:

display_field
Is the object of a SUM or COUNT display command.

sort_field
Is the object of a BY phrase.

Reference Usage Notes for WITHIN

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

Example Aggregating Values by Column

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

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

```
PAGE     1

                                                     PCT        PCT
                                                     SOLD       SOLD
                                                     WITHIN     WITHIN
   STORE_CODE       PROD_CODE          UNITS         TABLE      STORE

   ----------       ---------          -----         ------     ------
   K1               B10                13            1          30
                    B12                29            4          69

   14B              B10                60            8          15
                    B12                40            5          10
                    B17                29            4          7
                    C13                25            3          6
                    C7                 45            6          11
                    D12                27            3          7
                    E2                 80            11         21
                    E3                 70            10         18


PAGE     2

                                                     PCT        PCT
                                                     SOLD       SOLD
                                                     WITHIN     WITHIN
   STORE_CODE       PROD_CODE          UNITS         TABLE      STORE
   ----------       ---------          -----         ------     ------
   14Z              B10                30            4          18
                    B17                20            2          12
                    B20                15            2          9
                    C17                12            1          7
                    D12                20            2          12
                    E1                 30            4          18
                    E3                 35            5          21

   77F              B20                25            3          38
                    C7                 40            5          61
```

3 Viewing and Printing Report Output

Topics:

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

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

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

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

This chapter describes how to:

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

Displaying Reports in Hot Screen

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

This topic describes how to:

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

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

Activating Hot Screen

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

To check if Hot Screen is activated, issue

```
? SET
```

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

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

```
SET SCREEN=ON
```

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

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

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

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

Using PRINTPLUS

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

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

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

Note:

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

Syntax How to Use PRINTPLUS

Issue the command

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

Example Using PRINTPLUS With SUBFOOT and FOOTING

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

```
USE CAR FOCUS F
 END

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

The output is:

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

 SUMMARY OF CARS IN COUNTRY BY SEATING CAPACITY
 RELPMEK CAR SURVEY
```

Controlling the Display of Empty Reports

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

Issue this command from the command line

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

where:

`ON`

Generates an empty report when zero records are found.

`OFF`

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

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

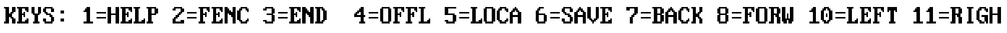
```
ON TABLE SET EMPTYREPORT ON
```

Note:

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

Accessing Help Information

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



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

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

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

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

Scrolling a Report

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

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

- Forward
- Backward
- Horizontally
- From fixed columns

Scrolling Forward

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

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

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

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

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

Scrolling Backward

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

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

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

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

Scrolling Horizontally

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

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

```
MORE =>
```

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

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

  ```
  LEFT n
  ```

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

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

  ```
  RIGHT n
  ```

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

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

  ```
  RESET
  ```

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

Scrolling From Fixed Columns (Fencing)

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

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

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

Scrolling Report Headings

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

To scroll report headings along with data the syntax is:

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

where:

`ON`

Enables BYSCROLL.

`OFF`

Disables BYSCROLL. OFF is the default.

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

Saving Selected Data

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

1. Position the cursor under the first character of the text to be saved.
2. Press PF6. FOCUS will save the text from that start character to the end of the line in a file with the file name SAVE. See Chapter 12, *Saving and Reusing Your Report Output*, for information about SAVE files.

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

Locating Character Strings

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

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

   ```
   ENTER STRING TO LOCATE /
   ```

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

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

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

```
LOCATE/string
```

Repeating Commands

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

For example:

```
RRIGHT 5
```

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

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

Redisplaying Reports

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

```
RETYPE
```

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

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

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

where:

field1

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

format1

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

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

Note:

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

```
TABLE FILE EMPLOYEE
.
.
.
END

RETYPE
RETYPE
```

Previewing Your Report

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

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

where:

`OFF`

Specifies that no retrieval is to be performed.

`ON`

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

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

Displaying BY Fields With Panels

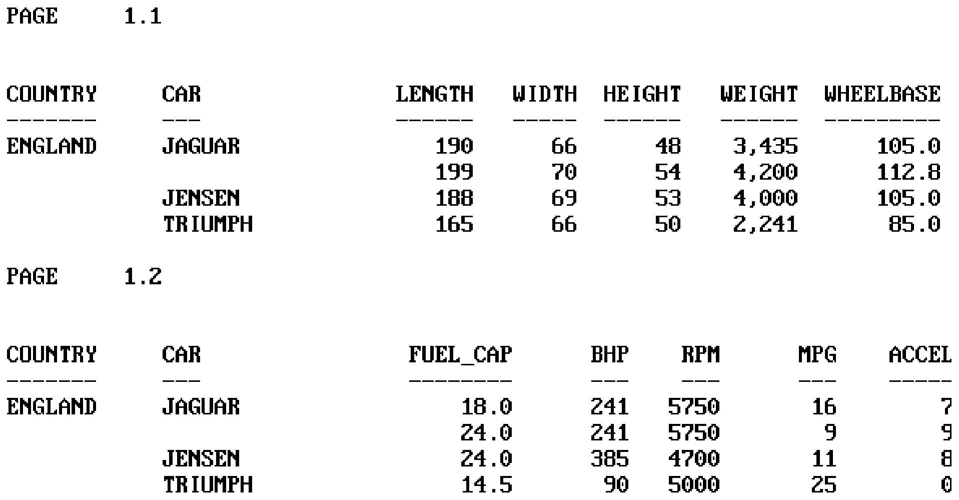
Hot Screen also enables you to display BY fields in the left portion of each panel of multi-panel reports. BY fields are vertical sort fields (see Chapter 4, *Sorting Tabular Reports*).The non-BY fields are displayed on the right portion of the panel. BY paneling is also available for OFFLINE reports.

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

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

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

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



```
PAGE     1.1


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

PAGE     1.2


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

BYPANEL Conditions

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

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

Scrolling by Columns of BY Fields in Panels

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

To move to the right one column, press PF10.

To move to the left one column, press PF11.

To move up within the same column, press PF7.

To move down within the same column, press PF8.

The SET COLUMNS Command

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

Note the following usage information:

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

Displaying Reports in the Panel Facility

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

To panel your report, issue

```
SET PANEL=n
```

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

For example:

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

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

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

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

Note:

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

Printing Reports

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

The OFFLINE Command

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

```
OFFLINE
```

before a report request.

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

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

```
OFFLINE
```

and then

```
RETYPE
```

at the FOCUS command line.

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

```
ON TABLE SET PRINT OFFLINE
```

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

You can reroute report output to your screen by issuing

```
ONLINE
```

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

```
OFFLINE CLOSE
```

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

Printing Reports in Hot Screen

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

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

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

1. Reformats the report to current page settings and then sends the entire report to a printer.

2. Prints the report page displayed, as formatted on the terminal screen.

3. Removes the print menu.

4. Creates a HOLD file using the entire report. The Master and FOCTEMP files will have the default name HOLD.

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

Displaying Reports in the Terminal Operator Environment

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

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

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

4 Sorting Tabular Reports

Topics:

- Sorting Rows: BY
- Sorting Columns: ACROSS
- Sorting Rows and Columns: Creating a Matrix
- Sorting More Efficiently: Using External Sorts
- Keeping Your Last Report Available Longer
- Specifying the Sort Order
- Grouping Numeric Data Into Ranges
- Selecting Sort Field Values by Rank: HIGHEST *n* / LOWEST *n*
- Ranking Sort Field Values: RANKED BY
- Hiding Sort Values: NOPRINT
- Using Multiple Display Commands in a Report Request

Sorting enables you to display the report information grouped in a particular order. You can organize the information by rows and columns.

The *sort field* controls the sequence of data items in the report and is used to organize rows and columns. Any field in the database can be the sort field. If you wish, you can select several sort fields, nesting one within another.

You sort the rows of a report using the BY phrase and the columns of a report using the ACROSS phrase. You can also create a matrix by sorting both rows and columns in the same report.

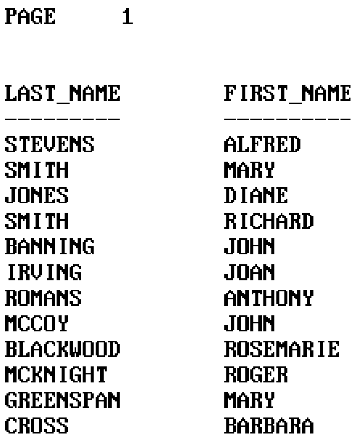
FOCUS provides you with additional sorting options, including the following:

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

Example Sorting Information by Columns

This request creates a report listing all employees.

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

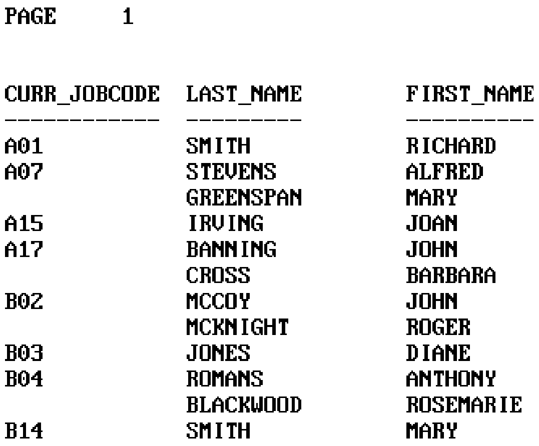


```
PAGE     1


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

However, you could sort the report by job code to display the employees in job code order.

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

The request produces this report.



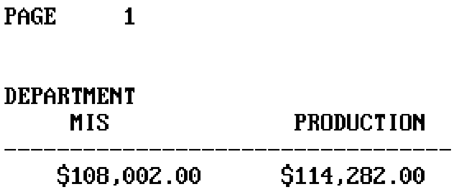
```
PAGE     1


CURR_JOBCODE  LAST_NAME        FIRST_NAME
------------  ---------        ----------
A01           SMITH            RICHARD
A07           STEVENS          ALFRED
              GREENSPAN        MARY
A15           IRVING           JOAN
A17           BANNING          JOHN
              CROSS            BARBARA
B02           MCCOY            JOHN
              MCKNIGHT         ROGER
B03           JONES            DIANE
B04           ROMANS           ANTHONY
              BLACKWOOD        ROSEMARIE
B14           SMITH            MARY
```

Example Sorting Information Across Rows

This request sums the employee salaries for each department listed in the database. The request sorts this information across a row.

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

The request produces this report.



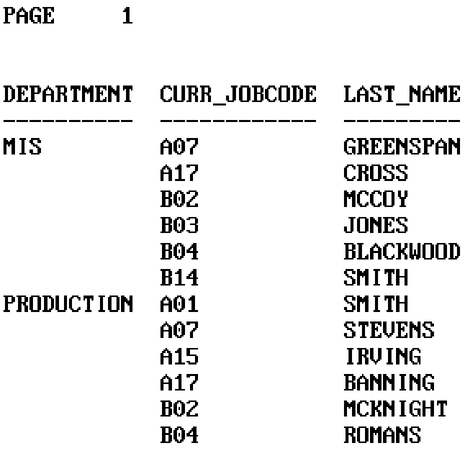
```
PAGE     1


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

Example Using Multiple Sort Fields

You can select several sort fields, nesting one within another, as with DEPARTMENT and CURR_JOBCODE in this example.

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

The request produces this report.



```
PAGE     1


DEPARTMENT  CURR_JOBCODE  LAST_NAME
----------  ------------  ---------
MIS         A07           GREENSPAN
            A17           CROSS
            B02           MCCOY
            B03           JONES
            B04           BLACKWOOD
            B14           SMITH
PRODUCTION  A01           SMITH
            A07           STEVENS
            A15           IRVING
            A17           BANNING
            B02           MCKNIGHT
            B04           ROMANS
```

Sorting Rows: BY

You can sort information down a report from top to bottom—that is, sort the rows of the report—using the BY phrase. You can include up to 32 BY phrases per report request (31 if using PRINT or LIST display commands).

Syntax

How to Sort by Rows

The syntax for the BY phrase is

```
BY sortfield
```

where:

sortfield

Is the name of the sort field.

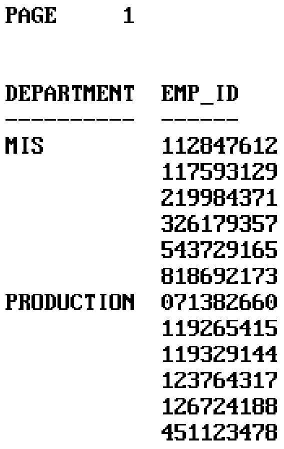
Example

Sorting Rows With BY

To print all employee IDs by department, you could issue the following request.

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

The request creates this report.



```
PAGE     1


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

Using Multiple Sort Fields With BY

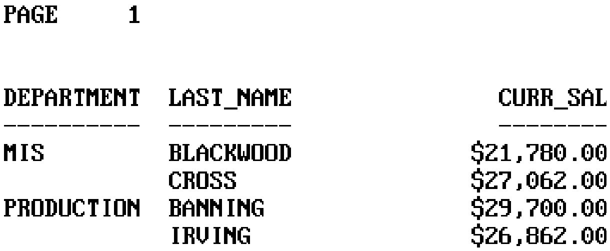
You can organize information in a report using more than one sort field. When several sort fields are used, the sequence of the BY phrases determines the sorting order: the first BY phrase sets the major sort break, the second BY phrase sets the second sort break, and so on. Each successive sort order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

Example Sorting With Multiple BYs

The following request uses multiple BYs to sort rows.

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

The request produces this report.



```
PAGE     1


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

Grouping and Aggregating Data Columns

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

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

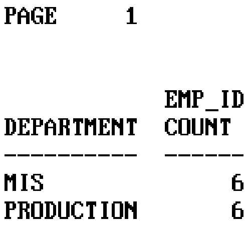
When you use the display commands PRINT and LIST, the report may generate several rows per sort value—specifically, one row for each occurrence of the display field. When you use the commands SUM and COUNT, the report generates one row for each sort value.

Example Grouping Columns With COUNT

This request counts instances of EMP_ID and sorts the count by department.

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

The request produces this report.



```
PAGE     1


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

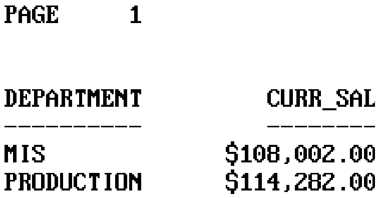
Example

Aggregating Columns With SUM

This request sums employee salaries (CURR_SAL) and sorts the values by department.

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

The request creates the following report.



```
PAGE     1


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

Reference

Usage Notes for Sorting Rows

- When using the display command LIST with a BY phrase, the LIST counter is reset to 1 each time the major sort value changes.
- Each sort field value appears only once in the report. For example, if there are six employees in the MIS department, a request that declares

  ```
  PRINT LAST_NAME BY DEPARTMENT
  ```

  will print MIS once, followed by six employee names.
- The default sort sequence is low-to-high character sorting sequence (a-z, A-Z, 0-9 for alphanumeric fields; 0-9 for numeric fields). You can specify other sorting sequences, as described in *Specifying the Sort Order* on page 4-20.
- You cannot use text fields as sort fields (text fields are those described in the Master File with a FORMAT value of TX).
- You cannot use a temporary field created by a COMPUTE command as a sort field. However, you can accomplish this indirectly by first creating a HOLD file that includes the field and then reporting from the HOLD file (HOLD files are described in Chapter 10, *Customizing Tabular Reports*). However, you can use a temporary field created by a DEFINE command, or by the DEFINE attribute in a Master File, as a sort field.
- If you specify several sort fields when reporting from a multi-path data source, all the sort fields must be in the same path.
- Sort phrases cannot contain format information for fields.

Sorting Columns: ACROSS

You can sort information across a report from left to right—that is, sort the columns—using the ACROSS phrase. You can have up to five ACROSS phrases per report request. Each ACROSS phrase can retrieve up to 95 sort field values. Your report can include up to 256 columns. The total number of ACROSS report columns is equal to the total number of ACROSS sort field values multiplied by the total number of display fields.

Syntax How to Sort Columns

The syntax for the ACROSS phrase is

```
ACROSS sortfield
```

where:

sortfield
Is the name of the sort field.

Reference Usage Notes for Sorting Columns

- Each sort field value is displayed only once in the report. For example, if there are six employees in the MIS department, a report that declares

  ```
  PRINT LAST_NAME ACROSS DEPARTMENT
  ```

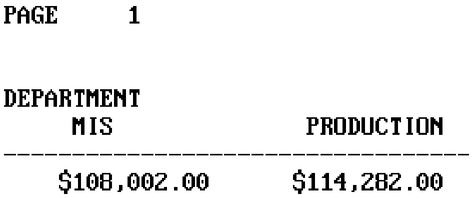
  will print MIS once, followed by six employee names.
- You cannot use text fields as sort fields (text fields are those described in the Master File with a FORMAT value of TX).
- You cannot use a temporary field created by a COMPUTE command as a sort field. However, you can accomplish this indirectly by first creating a HOLD file that includes the field and then reporting from the HOLD file (HOLD files are described in Chapter 12, *Saving and Reusing Your Report Output*). You can use a temporary field created by a DEFINE command, or by the DEFINE attribute in a Master File, as a sort field.
- For an ACROSS phrase, the SET SPACES command controls the distance between ACROSS sets. See Chapter 10, *Customizing Tabular Reports*, for more information.
- Sort phrases cannot contain format information for fields.
- If you specify several sort fields when reporting from a multi-path data source, all the sort fields must be in the same path.

Example Sorting Columns With ACROSS

To show each department's salary outlay, you could issue the following request.

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

The request produces this report.



```
PAGE     1


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

Using Multiple Sort Fields With ACROSS

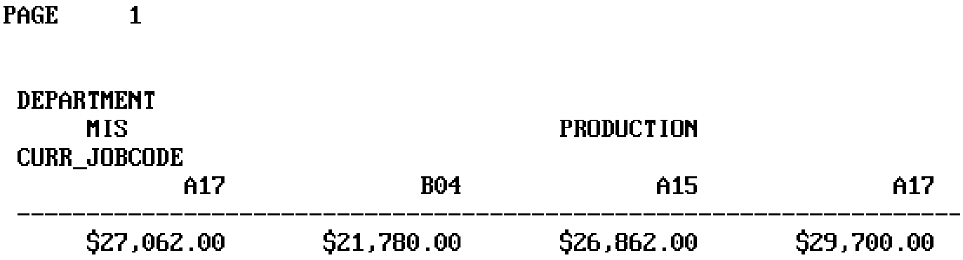
You can sort a report using more than one sort field. When several sort fields are used, the ACROSS phrase order determines the sorting order: the first ACROSS phrase sets the first sort break, the second ACROSS phrase sets the second sort break, and so on. Each successive sort order is nested within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

Example Sorting With Multiple ACROSS Phrases

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

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

The request creates this report.



```
PAGE     1


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

Grouping and Sorting Individual and Aggregate Data

There are two ways that you can sort information, depending upon which display command you use:

- You can sort and display all the individual values of a field by using the PRINT command.
- You can sort and display aggregate information, such as the number of field occurrences per sort value (by using the COUNT command), and the sum of field values (by using the SUM command).

Sorting Rows and Columns: Creating a Matrix

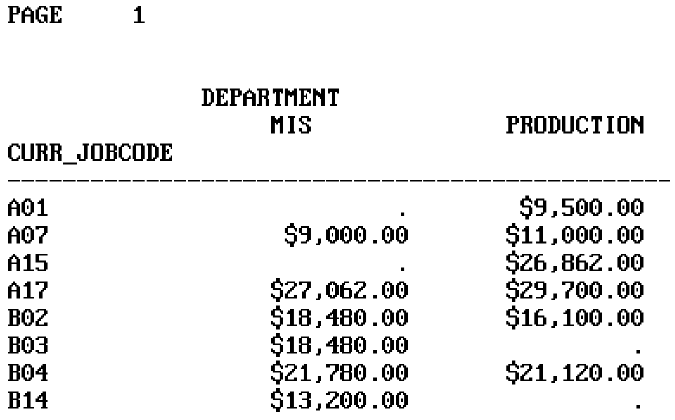
You can create a matrix report by sorting both rows and columns. When you include both BY and ACROSS phrases in a report request, information is sorted down the report and across the report, turning it into a matrix of information that you read like a grid. A matrix report can have several BY and ACROSS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

Example Creating a Simple Matrix

The following request displays total salary outlay across departments and by job codes.

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

The request produces this report.



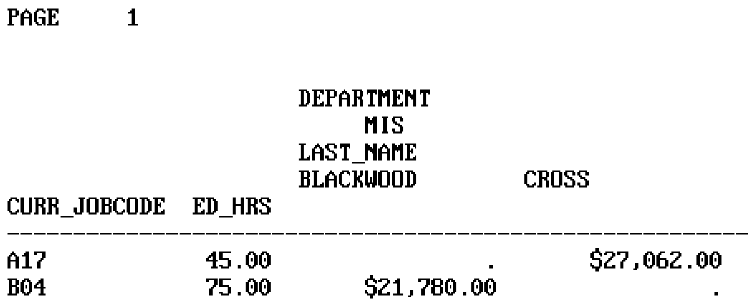
```
PAGE     1


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

Example Creating a Matrix With Several Sort Fields

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

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

The request produces this report.



```
PAGE     1


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

Sorting More Efficiently: Using External Sorts

When FOCUS generates a report, by default it sorts the report using its own sorting procedure. This internal FOCUS sort is optimized for reports of up to approximately 5000 records (for example, up to 5000 lines of a tabular report).

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

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

Requirements

You can use an external sort with any TABLE, EMR, MATCH, and GRAPH report request. You can use the following sort products:

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

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

  It is recommended that you issue a GLOBAL TXTLIB command to identify the location of the sort software. If FOCUS cannot locate the sort, it issues a GLOBAL TXTLIB SORTLIB command. If FOCUS still cannot locate the sort software, it will terminate with the following error message:

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

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

FOCUS always respects your sort product’s installation parameters.

How FOCUS Determines Which Sort to Use

FOCUS determines whether or not to use an external sort by evaluating the following criteria in this sequence:

1. **BINS.** If FOCUS can sort an entire report within its work area (BINS), it does so and does not invoke the external sort, even if the SET EXTSORT command is set on.
2. **EXTERNAL.** If BINS is not large enough to sort the entire report and the SET EXTSORT command is on, FOCUS sorts approximately the first 5000 records and passes remaining records to the external sort utility.

Controlling External Sorting: SET EXTSORT

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

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

where:

`ON`

Enables FOCUS to use a dedicated external sorting product to sort reports. This is the default.

`OFF`

Uses the FOCUS internal sorting procedure to sort all FOCUS reports.

To determine which sort FOCUS used for a given report, issue the ? STAT command following the report request. The command displays the following values for the SORT USED parameter:

FOCUS

FOCUS used its internal procedure to sort the entire report.

SQL

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

EXTERNAL

An external sorting product sorted the report.

FOCUS report optimization determines the scope of the external sort. FOCUS sorts the first group of approximately 5000 records, and the external sort handles the remainder.

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

NONE

The report did not require sorting.

Aggregation by External Sort

External sorts can be used to perform aggregation with a significant decrease in processing time in comparison to using the sort facility of FOCUS. The gains will be most notable with relatively simple requests against large databases.

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

Syntax How to Use Aggregation in Your External Sort

```
SET EXTAGGR = aggropt
```

where:

`aggropt`

Can be one of the following:

`OFF` disallows aggregation by an external sort.

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

`ON` allows aggregation by an external sort. ON is the default.

Reference Usage Notes for Aggregating With an External Sort

- You must be using SYNCSORT or DFSORT.
- EXTAGGR cannot be set to OFF.
- Your query should be simple. (AUTOTABLEFable)
- The PRINT display command may not be used in the query.
- SET ALL must be equal to OFF.
- Only the following column prefixes are allowed: SUM, AVG, CNT, FST.
- Columns can be COMPUTEd or have a ROW-TOTAL.
- CMS DFSORT does not support aggregation of numeric data types. When SET EXTAGGR = NOFLOAT and your query aggregates numeric data, the external sort is not called and aggregation is performed by the FOCUS sort.

Example Using an External Sort for Aggregation to 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. SUMPREFIX is described in *Changing Retrieval Order With Aggregation* on page 4-16.

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.

HOLD From External Sort

External sorts can be used to create HOLD files. This can lead to savings of up to twenty percent in 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.

Reference Usage Notes 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 data set named FOCSORT. Its default is ON so that performance gains may be realized.
- SET ALL must be OFF.
- There cannot be an IF/WHERE TOTAL or BY TOTAL in the request.
- If a request contains a SUM command, EXTAGGR must be set ON and the only column prefixes allowed are SUM and FST.
- If a request contains a PRINT command, the column prefixes allowed are SUM, AVE, MAX, MIN, FST and LST.

Estimating SORTWORK Sizes for an External Sort

Specifying the size of your SORTWORK files is usually done when you install an external sort product. Large FOCUS queries that use external sorts can rapidly exhaust SORTWORK space, resulting in FOC909 errors. This problem is solved if the sorts include the parameter 'FILSZ=E*n*' when invoked. This parameter enables the sorting algorithms to estimate SORTWORK space requirements for each sort parameter request.

ESTRECORDS is used to pass the estimated number of records to be sorted in the request. In order to make an accurate estimate for your ESTRECORDS setting, it is suggested that you run the report without an external sort in order to get a record count.

Syntax How to Provide an Estimate of Input Records for Sorting

The syntax is

```
ON TABLE SET ESTRECORDS n
```

where:

n

Is the estimated number of records to be sorted.

Note:

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

  ```
  SET ESTRECORDS = n
  ```

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

Displaying External Sort Messages

By default, FOCUS does not display any messages created by your external sort product. However, you may wish to display these messages on your screen for diagnostic purposes. The following instructions tell you how to display messages for DFSORT and SyncSort under MVS.

To display DFSORT messages:

1. Create a sequential file with these attributes:

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

2. Create the following record beginning in column 2

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

   where *trace_name* is any user-defined ddname.

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

To display SyncSort messages:

1. Create a sequential file with these attributes:

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

2. Create the following record beginning in column 2:

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

   where *trace_name* is any user-defined ddname.

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

Keeping Your Last Report Available Longer

The internal matrix is created with each TABLE, EMR, MATCH, and GRAPH request when the SET SAVEMATRIX feature is activated.

The internal matrix may be saved using the SET SAVEMATRIX command

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

where:

ON
: Saves the internal matrix from the last report request, preventing it from being overwritten. This is the default.

OFF
: Overwrites the internal matrix for each request.

The TABLE report, EMR report, MATCH request, or graph is available for the duration of your FOCUS session, or until you generate a new report or graph which overwrites it. While a report (or graph) is available, you can:

- Redisplay it using the RETYPE or REPLOT command.
- Extract and save data from it using the HOLD, SAVE, and SAVB commands.

For example, you may create a report, then execute a procedure that contains a Dialogue Manager command (which would otherwise overwrite the internal matrix), and recall the report using the RETYPE command:

```
TABLE FILE EMPLOYEE
.
.
.
END
EX DMFEX
RETYPE
```

Note: SET SAVEMATRIX is not available with the TABLEF command.

Reference 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 the *Developing Applications* manual.

Specifying the Sort Order

FOCUS automatically displays sort field values in ascending order, beginning with the lowest value and continuing to the highest value. The default sorting sequence is the low-to-high character sorting sequence (a-z, A-Z, 0-9 for alphanumeric fields; 0-9 for numeric fields).

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

Syntax How to Specify the Sort Order

The syntax is

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

where:

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

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

`sortfield`
: Is the name of the sort field.

Example Sorting in Ascending Order

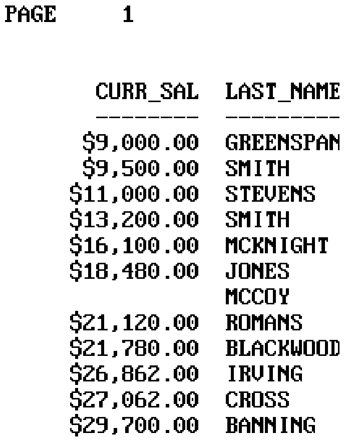
The following report request does not specify a particular sorting order, and so, by default, it lists salaries ranging from the lowest to the highest.

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

You can also explicitly specify this same ascending order by using the keyword LOWEST.

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

Either request produces the report shown below.



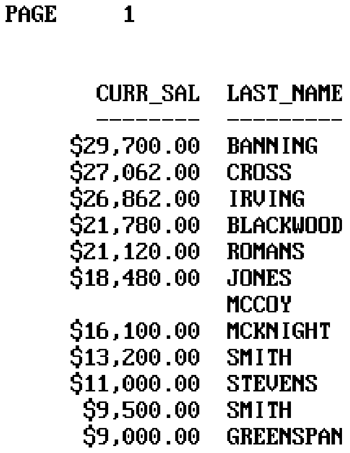
```
PAGE     1


       CURR_SAL  LAST_NAME
       --------  ---------
      $9,000.00  GREENSPAN
      $9,500.00  SMITH
     $11,000.00  STEVENS
     $13,200.00  SMITH
     $16,100.00  MCKNIGHT
     $18,480.00  JONES
                 MCCOY
     $21,120.00  ROMANS
     $21,780.00  BLACKWOOD
     $26,862.00  IRVING
     $27,062.00  CROSS
     $29,700.00  BANNING
```

Example Sorting in Descending Order

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

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

The request produces this report.



```
PAGE     1


      CURR_SAL  LAST_NAME
      --------  ---------
    $29,700.00  BANNING
    $27,062.00  CROSS
    $26,862.00  IRVING
    $21,780.00  BLACKWOOD
    $21,120.00  ROMANS
    $18,480.00  JONES
                MCCOY
    $16,100.00  MCKNIGHT
    $13,200.00  SMITH
    $11,000.00  STEVENS
     $9,500.00  SMITH
     $9,000.00  GREENSPAN
```

Specifying Your Own Sort Order

FOCUS automatically displays sort field values in ascending order, beginning with the lowest value and continuing to the highest value. The default sorting sequence is the low-to-high character sorting sequence (a-z, A-Z, 0-9 for alphanumeric fields; 0-9 for numeric fields).

You can also override this default and display values in your own user-defined sorting sequence.

You can define the order in which you want a report to be sorted. To do this, you need to decide the following:

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

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

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

Syntax

How to Define Your Own Sort Order

The syntax for using BY ROWS OVER is

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

where:

sortfield
Is the name of the sort field.

value1
Is the sort field value that is first in the sorting sequence.

value2
Is the sort field value that is second in the sorting sequence.

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

An alternative syntax is

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

which uses the row-based reporting phrase FOR, described in Chapter 17, *Creating Extended Matrix Reports*.

Reference

Usage Notes for Defining Your Sort Order

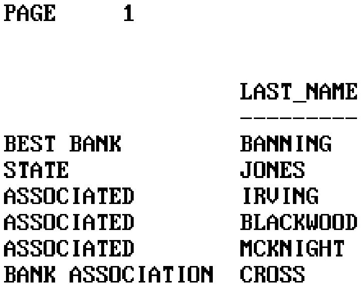
- Any sort field value that you do not specify in the BY ROWS OVER phrase is not included in the sorting sequence and does not appear in the report.
- Sort field values that contain embedded blank spaces should be enclosed in single quotation marks.
- Any sort field value that you do specify in the BY ROWS OVER phrase is included in the report, whether or not there is data.
- The name of the sort field is not included in the report.
- Each report request can contain one BY ROWS OVER phrase.

Example Defining Your Row Sort Order

To sort employees by the bank at which their paycheck is automatically deposited, and to define your own label within the sorting sequence for the bank field, issue the following request:

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

The request generates this report.



```
PAGE     1


                  LAST_NAME
                  ---------
BEST BANK         BANNING
STATE             JONES
ASSOCIATED        IRVING
ASSOCIATED        BLACKWOOD
ASSOCIATED        MCKNIGHT
BANK ASSOCIATION  CROSS
```

Syntax How to Define Column Sort Sequence

The syntax for using ACROSS COLUMNS AND is

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

where:

sortfield
: Is the name of the sort field.

value1
: Is the sort field value that is first in the sorting sequence.

value2
: Is the sort field value that is second in the sorting sequence.

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

Reference Usage Notes for Defining Column Sort Sequence

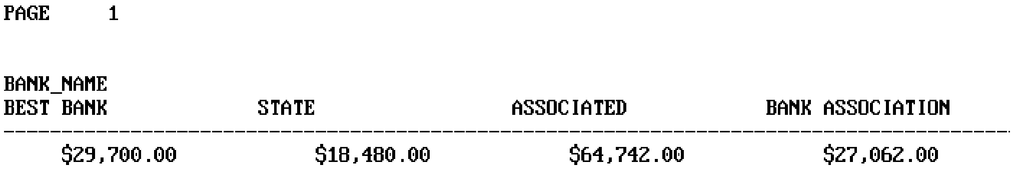
- Any sort field value that you do not specify in the ACROSS COLUMNS AND phrase is not included in the label within the sorting sequence and does not appear in the report.
- Sort field values that contain embedded blank spaces should be enclosed in single quotation marks.
- Any sort field value that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

Example Defining Column Sort Sequence

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

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

The request produces the report below.



```
PAGE     1


BANK_NAME
BEST BANK            STATE                ASSOCIATED           BANK ASSOCIATION
------------------------------------------------------------------------------------
     $29,700.00           $18,480.00           $64,742.00           $27,062.00
```

Grouping Numeric Data Into Ranges

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

There are two ways of defining groups:

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

The FOR phrase is usually used to produce matrix reports and is part of the Extended Matrix Reporting (EMR) facility. However, you can also use it to create columnar reports that group sort field values in unequal ranges.

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

The FOR phrase is described in more detail in Chapter 17, *Creating Extended Matrix Reports*.

Syntax How to Define Groups of Equal Range

The syntax for using IN-GROUPS-OF is

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

where:

sortfield
Is the name of the sort field. The sort field must be numeric—that is, its format must be I, F, D, or P.

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

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

Reference Usage Notes for Defining Groups of Equal Range

There are several usage considerations:

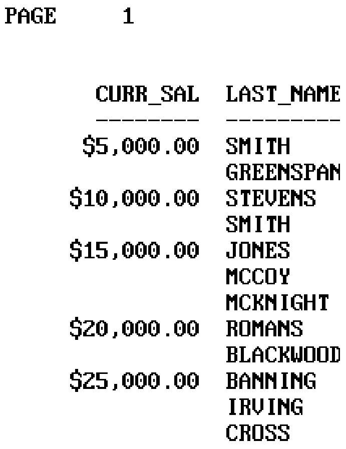
- The first sort field range starts from zero.
- Each report request can contain up to five IN-GROUPS-OF phrases.
- The IN-GROUPS-OF phrase can only be used once per BY field.
- The value displayed is the end point of each range.

Example Defining Groups of Equal Ranges

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

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

The request generates this report which shows LAST_NAME in groups of 5000:



```
PAGE     1


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

Syntax How to Define Custom Groups of Data Values

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

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

where:

sortfield
Is the name of the sort field.

begin
Is a value that identifies the beginning of a range.

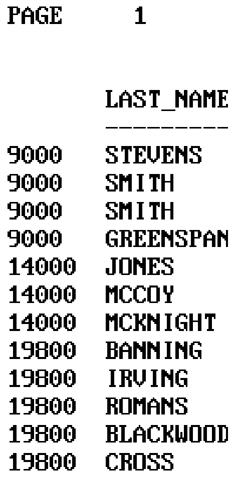
end
Is a value that identifies the end of a range.

Example Defining Custom Groups of Data Values

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

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

The request produces the following report:



```
PAGE     1


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

Selecting Sort Field Values by Rank: HIGHEST *n* / LOWEST *n*

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

Syntax

How to Rank Sort Field Values

The syntax is

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

where:

`HIGHEST n`

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

`LOWEST n`

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

`sortfield`

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

Reference

Usage Notes for Ranking Sort Fields

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

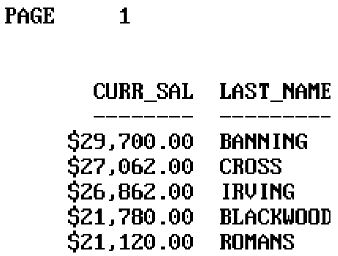
Example

Restricting Sort Field Values to a Group

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

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

The request produces this report:



```
PAGE     1


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

Ranking Sort Field Values: RANKED BY

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

Syntax How to Rank Sort Field Values

The syntax is

```
RANKED BY sortfield
```

where:

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

Reference Usage Notes for Ranking Sort Field Values

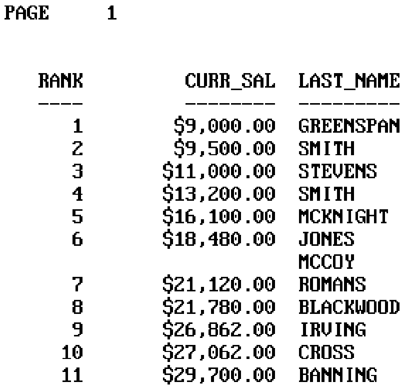
- You can replace the RANK column heading with one that you define by using the AS phrase, as described in Chapter 10, *Customizing Tabular Reports*.
- Several report rows will have the same rank if they have identical sort field values.

Example Ranking Sort Field Values

To display a list of employee names in salary order, showing the rank of each employee by salary, issue the following request:

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

The request creates this report:



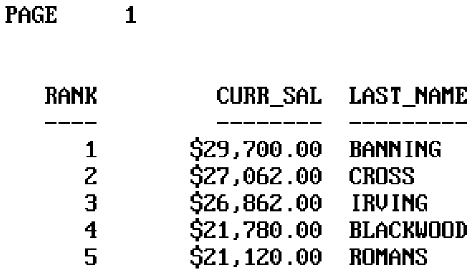
```
PAGE     1


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

Example Ranking and Restricting Sort Field Values

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

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

This request generates this report:



```
PAGE     1


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

Hiding Sort Values: NOPRINT

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

Syntax How to Hide Sort Values

The syntax is

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

where:

`sortfield`

Is the name of the sort field.

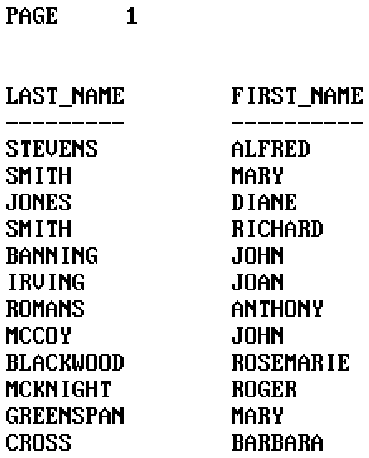
You can use SUP-PRINT as a synonym for NOPRINT.

Example Hiding Sort Values

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

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

The request produces this report, which lists the names in the order that they were entered into the database:

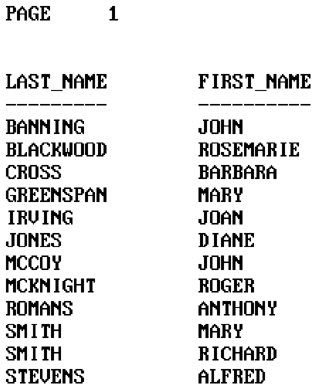


```
PAGE     1


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

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

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

This request provides the desired result:



```
PAGE     1


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

Using Multiple Display Commands in a Report Request

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

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

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

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

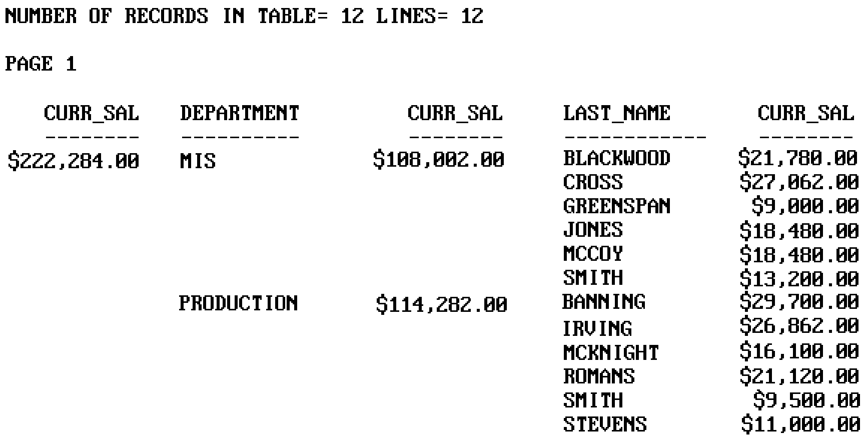
Example Using Multiple Display and Sort Fields

The following request summarizes several levels of detail in the database:

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

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

This request produces the following report:



```
NUMBER OF RECORDS IN TABLE= 12 LINES= 12

PAGE 1

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

5 Selecting Records for Your Report

Topics:

- Choosing a Filtering Method (WHERE or IF)
- Selections Based on Individual Values: WHERE
- Selection Based on Aggregate Values: WHERE TOTAL
- Types of WHERE Selection Tests
- Selecting Records Using IF Phrases
- Types of IF Selection Tests
- Assigning Screening Conditions to a File

When generating a report and specifying which fields are to be displayed, you may not want to display every instance of a field. By including additional selection criteria, you can display only those field values that meet your needs.

In effect, you can select a subset of the data—a subset that you can easily redefine each time you issue the report request.

You can even base record selection on whether or not a field has a null value, falls within a range of values, or does not contain a certain value. You can specify selection criteria directly in the report request, or in a separate file accessed by many report requests.

You can also select records based on the aggregate value of a field (for example, on the sum of a field's values, or on the average of a field's values). For non-FOCUS data files that have group keys, you can select records based on group key values.

In addition to these selection methods, you can limit the records included in a report not by the value of a field, but simply by the number of records—for example, by including only the first 50 records.

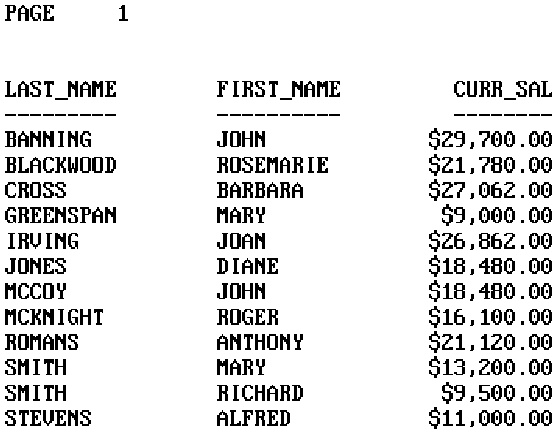
The process of selecting records is sometimes referred to as filtering or screening, because the records that are not selected are filtered out of the report. You can store your screening conditions in a separate file for use by multiple report requests. See *Assigning Screening Conditions to a File* on page 5-29.

Example Using Simple Selection Criteria

For example, the following report request prints the names and salaries of all employees:

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

The report that results is shown below.



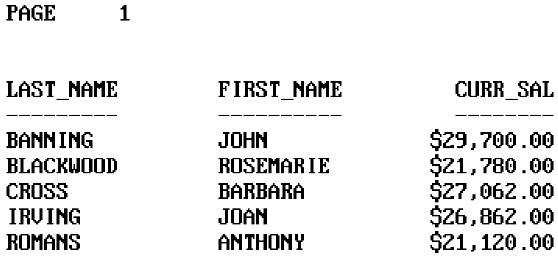
```
PAGE     1


LAST_NAME        FIRST_NAME         CURR_SAL
---------        ----------         --------
BANNING          JOHN             $29,700.00
BLACKWOOD        ROSEMARIE        $21,780.00
CROSS            BARBARA          $27,062.00
GREENSPAN        MARY              $9,000.00
IRVING           JOAN             $26,862.00
JONES            DIANE            $18,480.00
MCCOY            JOHN             $18,480.00
MCKNIGHT         ROGER            $16,100.00
ROMANS           ANTHONY          $21,120.00
SMITH            MARY             $13,200.00
SMITH            RICHARD           $9,500.00
STEVENS          ALFRED           $11,000.00
```

To show only the names and salaries of employees earning more than $20,000 a year, issue essentially the same request, but add criteria to select only database records for those higher-salaried employees, as shown below.

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

In this example, CURR_SAL is a selected field, and CURR_SAL GT 20,000 (current salary greater than $20,000) is the selection criterion. FOCUS retrieves only those records with a CURR_SAL value greater than 20,000; it ignores all other records.

The report that results is shown below:



```
PAGE     1


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

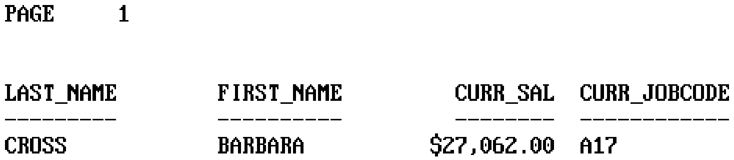
Example Using Complex Selection Criteria

You can also apply more complex selection criteria to your report requests. For example, the following request selects only those employees who:

- Earn more than $20,000.
- Work in the MIS department.
- Have an “A” job code:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME AND CURR_SAL AND CURR_JOBCODE
BY LAST_NAME NOPRINT
WHERE (CURR_SAL GT 20000) AND (DEPARTMENT IS 'MIS')
       AND (CURR_JOBCODE CONTAINS 'A')
END
```

This request produces the following report:



```
PAGE     1


LAST_NAME        FIRST_NAME         CURR_SAL  CURR_JOBCODE
---------        ----------         --------  ------------
CROSS            BARBARA          $27,062.00  A17
```

Choosing a Filtering Method (WHERE or IF)

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

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

Selections Based on Individual Values: WHERE

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

Syntax How to Select Records With WHERE

The basic selection syntax using WHERE is

```
WHERE criteria[;]
```

where:

`criteria`

Are the criteria for selecting records to include in the report. The criteria must be a valid FOCUS expression that evaluates as true or false (that is, a Boolean expression). Expressions are described in detail in Chapter 8, *Using Expressions*. Operators that can be used in WHERE expressions—such as CONTAINS, IS, and GT—are described in *Types of WHERE Selection Tests* on page 5-10.

`;`

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

Reference Usage Notes for WHERE Phrases

The WHERE phrase can include:

- Most expressions that would be valid on the right-hand side of a DEFINE. However, the logical expression IF … THEN … ELSE cannot be used.
- Real fields, temporary fields, fields in joined files, and fields with prefix operators (WHERE TOTAL only).
- The operators EQ, NE, GE, GT, LT, LE, CONTAINS, OMITS, FROM … TO, NOT-FROM … TO, INCLUDES, EXCLUDES, LIKE, and NOT LIKE.
- All arithmetic operators (+, -, *, /, **), as well as functions (MIN, MAX, ABS, and SQRT, for example).
- An alphanumeric expression which can be a literal, or a function yielding an alphanumeric result using EDIT or DECODE.

  Note: Files used with DECODE expressions can contain two columns, one for field values and one for numeric decode values.
- All literals—including date literals, but not numeric literals—enclosed in single quotation marks.
- All functions and subroutines.

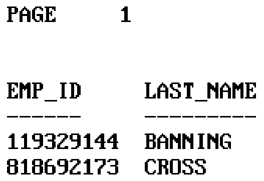
You can build complex selection criteria by joining simple expressions with AND and OR logical operators and, optionally, adding parentheses to explicitly specify the order of evaluation. This is easier than trying to achieve the same effect with the IF phrase, which would require the use of a separate DEFINE command.

Example Using Multiple WHERE Phrases

You can use as many WHERE phrases as necessary to define your selection criteria. Multiple WHERE phrases are treated as though they are connected with AND as long as they are in a single path. For example, this request uses multiple WHERE phrases.

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

The request produces the following report:



```
PAGE     1


EMP_ID     LAST_NAME
------     ---------
119329144  BANNING
818692173  CROSS
```

Reading Selection Criteria From a File

Instead of typing literal test values in a WHERE phrase, you can store them in a file and simply reference the file. You can then select records based on equality—or inequality—tests on values stored in the file. This method has the following advantages:

- It saves time by specifying large sets of selection values just once, and ensures consistency by maintaining it in just one location, yet it can be referenced in as many report requests as you wish.
- If the selection values already exist in a data source, you can quickly create a file of selection values by generating a report and saving the output in an extract file—a HOLD or SAVE file. You can then read selection values from the extract file.

  If you use a HOLD file, it must either be in internal format (the default), or in simple character (ALPHA) format; if you use a SAVE file, it must be in simple character format (the default). You can also use a SAVB file if the selection values are alphanumeric. See Chapter 12*, Saving and Reusing Your Report Output,* for information on HOLD, PCHOLD, SAVE, and SAVB files. Note that on MVS a HOLD file in internal format that is used for selection values must be allocated to ddname HOLD (the default); the other types of extract files used for this purpose can be allocated to any ddname.
- Allows mixed case and special characters.

Syntax How to Read Selection Criteria From a File

The syntax for using WHERE to select records based on values in a file is

```
WHERE [NOT] fieldname IN FILE file
```

where:

`NOT`

Selects records in which the specified field's value is not found in the file.

`fieldname`

Is the name of the selection field. It can be any real or temporary field in the database being reported on.

`file`

Is the name of the file. For MVS, this is the ddname assigned by a FOCUS DYNAM or TSO ALLOCATE command. On CMS, the ddname is assigned by a FILEDEF command. The file can contain up to 32,767 literals.

Reference Usage Notes for Reading Selection Criteria From a File

- Each value in the file must be on a separate line.
- The first value must start in column one.
- The values are assumed to be in character format, unless the file name is HOLD, and numeric digits are converted to internal computational numbers where needed (that is, binary integer, and so forth).
- Alphanumeric values with embedded blanks or any mathematical operator (-, +, *, /) must be enclosed in single quotation marks.
- When a compound WHERE phrase uses IN FILE more than once, the specified files must have the same record formats.
- If your list of literals is too large for the 32,767 literals restriction, an error is displayed.

Example Reading Selection Criteria From a File

For example, this request uses selection criteria from the file EXPER that contains the values B10 and B12. FOCUS will select all records for which PROD_CODE has a value of B10 or B12:

```
TABLE FILE PROD
SUM UNIT_COST
BY PROD_NAME
WHERE PROD_CODE IN FILE EXPER
END
```

If you entered selection criteria directly in the request, the WHERE phrase would be:

```
WHERE PROD_CODE EQ 'B10' or 'B12'
```

Selections Based on Group Key Values

Certain types of non-FOCUS data files use group keys. A group key is a single key composed of several fields, each of which may be in a different internal format. FOCUS supports the use of a group name to refer to a group key’s fields.

To select records based on a group key value, you need to supply the values of each field. The values must be separated by a slash character (/). A WHERE phrase which refers to a group field cannot be used in conjunction with AND or OR.

Example Selecting Records Using Group Keys

For example, a data file has a group key named PRODNO which contains three separate fields. The first is stored in alphanumeric format, the second as a packed decimal, the third as an integer. A screening phrase on this group might be:

```
WHERE PRODNO EQ 'RS/62/83'
```

For further details on working with non-FOCUS data files, see the *Describing Data* manual.

Selection Based on Aggregate Values: WHERE TOTAL

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

In WHERE tests, data is evaluated before it is retrieved. In WHERE TOTAL tests, however, data is selected after all the data has been retrieved and processed.

Syntax How to Select Records With WHERE TOTAL

The syntax of the WHERE TOTAL phrase is

```
WHERE TOTAL criteria[;]
```

where:

criteria

Are the criteria for selecting records to include in the report. The criteria must be a valid FOCUS expression that evaluates as true or false (that is, a Boolean expression). Expressions are described in detail in Chapter 8, *Using Expressions*. Operators that can be used in WHERE expressions—such as IS and GT—are described in *Types of WHERE Selection Tests* on page 5-10.

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

Reference Usage Notes for WHERE TOTAL

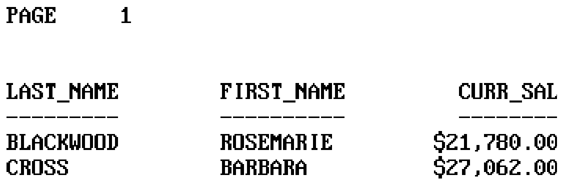
- Any reference to a computed field, or use of a feature that aggregates values, such as TOT.field, AVE.field, and so forth, requires use of WHERE TOTAL.
- WHERE TOTAL tests are performed at the lowest sort level.
- Alphanumeric and date literals must be enclosed within single quotation marks.
- When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

Example Using WHERE TOTAL With WHERE

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

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

The request generates this report:



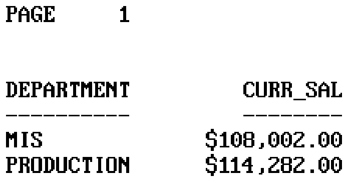
```
PAGE     1


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

Example Using WHERE TOTAL

The following example sums current salaries by department.

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

This request creates the following report:

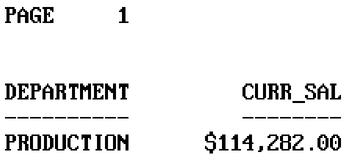


```
PAGE     1


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

When you add WHERE TOTAL to the request, the request lists only the departments where the total of the salaries is more than 110,000. The values for each department are calculated and then each final value is compared to 110,000.

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

This request produces the following report.



```
PAGE     1


DEPARTMENT        CURR_SAL
----------        --------
PRODUCTION     $114,282.00
```

Types of WHERE Selection Tests

You can define WHERE selection criteria using the following operators.

Operator	**Meaning**
`EQ` `IS`	Tests for and selects values equal to the test expression.
`NE` `IS-NOT`	Tests for and selects values not equal to the test expression.
`GE`	Tests for and selects values greater than or equal to the test value (based on the characters 0 to 9 for numeric values, A to Z and a to z for alphanumeric values).

`GT`
`EXCEEDS`
`IS-MORE-THAN`
	Tests for and selects values greater than the test value.

`LT` `IS-LESS-THAN`	Tests for and selects values less than the test value.
`LE`	Tests for and selects values less than or equal to the test value.
GE *lower* AND ... LE *upper*	Tests for and selects values within a range of values.
LT *lower* OR ... GT *upper*	Tests for and selects values outside of a range of values.
FROM *lower* TO *upper*	Tests for and selects values within a range of values.
IS-FROM *lower* TO *upper*	Tests for and selects values within a range of values. This is alternate syntax for FROM lower to UPPER; both operators produce identical results.
NOT-FROM *lower* TO *upper*	Tests for and selects values that are outside a range of values.

Operator	**Meaning**

`IS MISSING`
`IS-NOT MISSING`
`NE MISSING`
	Tests whether a field contains null values—that is, if some instances of the field contain no data (have missing data). For information on missing data, see Chapter 13, *Handling Records With Missing Field Values*.

`CONTAINS` `LIKE`	Tests for and selects values that include a character string matching test value. The string can occur in any position in the value being tested. When used with WHERE, it can test alphanumeric fields; when used with IF, it can test both alphanumeric and text fields.

`OMITS`
`NOT LIKE`
	Tests for and selects values that do not include a character string matching test value. The string cannot occur in any position in the value being tested. When used with WHERE, it can test alphanumeric fields; when used with IF, it can test both alphanumeric and text fields.

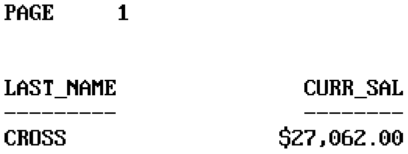
`INCLUDES`	Tests whether a chain of values of a given field in a child segment includes all of a list of literals.
`EXCLUDES`	Tests whether a chain of values of a given field in a child segment excludes all of a list of literals.
`IN (`*z,x,y*`)`	Selects records based on values found in an unordered list.
`NOT ... IN (`*z,x,y*`)`	Selects records based on values not found in an unordered list.
`IN FILE`	Selects records based on values stored in a sequential file.
`NOT ... IN FILE`	Selects records with field values not found in a sequential file.

Using Compound Logical Expressions

While FOCUS always evaluates logical ANDs before logical ORs, it will be easier for you to correctly develop and read compound logical expressions if you put parentheses around each simple expression. This is especially true when mixing literal OR tests with logical ANDs and ORs, as in the following example.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY LAST_NAME
WHERE (LAST_NAME EQ 'CROSS' OR 'JONES')
AND (CURR_SAL GT 22000)
END
```

The request produces the following report:



```
PAGE     1


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

Comparing a Field to One or More Values

The following examples illustrate field selection criteria that use one or more values. You may use the operators: EQ, IS, IS-NOT, EXCEEDS, IS-LESS-THAN, and IN.

Example 1: The field LAST_NAME must equal the value JONES:

```
WHERE LAST_NAME EQ 'JONES'
```

Example 2: The field LAST_NAME begins with ‘CR’ or ‘MC:’

```
WHERE EDIT (LAST_NAME, '99') EQ 'CR' OR 'MC'
```

Example 3: The field AREA must not equal the value EAST or WEST:

```
WHERE AREA IS-NOT 'EAST' OR 'WEST'
```

Example 4: The field AREA must equal the field named REGION:

```
WHERE AREA EQ REGION
```

Example 5: The ratio between retail cost and dealer cost must be greater than 1.25:

```
WHERE RETAIL_COST/DEALER_COST GT 1.25
```

Example 6: The field UNITS must be equal to or less than the value 50, and AREA must not be equal to either NORTH EAST or WEST. Note the use of single quotation marks around NORTH EAST. Strings that have an embedded space (in this example, the space between NORTH and EAST) must be enclosed within single quotation marks.

```
WHERE UNITS LE 50 WHERE AREA IS-NOT 'NORTH EAST' OR 'WEST'
```

Example 7: The value of AMOUNT must be greater than 40:

```
WHERE AMOUNT EXCEEDS 40
```

Example 8: The value of AMOUNT must be less than 50:

```
WHERE AMOUNT IS-LESS-THAN 50
```

Example 9: The value of SALES must be equal to one of the numeric values in the unordered list. Use commas or blanks to separate the list values.

```
WHERE SALES IN (43000,12000,13000)
```

Example 10: The value of CAR must be equal to one of the alphanumeric values in the unordered list. Single quotation marks must enclose alphanumeric list values.

```
WHERE CAR IN ('JENSEN','JAGUAR')
```

Specifying Range Tests With FROM and TO

Use the operators FROM … TO and NOT-FROM … TO, to determine whether field values fall within or outside a given range. You can use either values or expressions to specify the lower and upper boundaries. Range tests can also be applied on the sort control fields. The range test is specified immediately after the sort phrase.

Syntax How to Specify a Range Test

The syntax for the range test is

```
WHERE [TOTAL] fieldname {FROM|IS-FROM} lower TO upper
WHERE [TOTAL] fieldname  NOT-FROM      lower TO upper
```

where:

fieldname
  Is any valid field name or alias.

FROM|IS-FROM
  Is used to determine if field values are within a given range.

NOT-FROM
  Is used to determine if field values are outside a given range.

lower TO *upper*
  Are numeric or alphanumeric values or expressions that indicate lower and upper boundaries. You may add parentheses around expressions for readability.

Example Range Test With FROM ... TO

An example of a range test using expressions as boundaries follows:

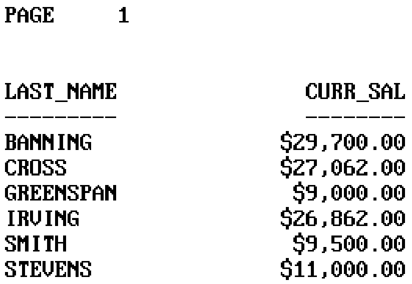
```
WHERE SALES FROM (DEALER_COST * 1.4) TO (DEALER_COST * 2.0)
```

Example Range Test With NOT-FROM ... TO

This request displays only employees whose salaries do not fall between $12,000 and $22,000:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY LAST_NAME
WHERE CURR_SAL NOT-FROM 12000 TO 22000
END
```

The request produces the following report:



```
PAGE     1


LAST_NAME              CURR_SAL
---------              --------
BANNING              $29,700.00
CROSS                $27,062.00
GREENSPAN             $9,000.00
IRVING               $26,862.00
SMITH                 $9,500.00
STEVENS              $11,000.00
```

Example Range Tests on Sort Fields

For example

```
BY MONTH FROM 4 TO 8
```

or:

```
ACROSS MONTH FROM 6 TO 10
```

Specifying Range Tests With GE and LE

The operators GE (Greater Than or Equal) and LE (Less Than or Equal) can also be used to specify a range. GE … LE enables you to specify values as the range test boundaries.

Syntax

How to Specify Range Tests With GE and LE

To select values that fall within a range, use AND:

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

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

```
WHERE fieldname LT lower OR fieldname GT upper
```

Example

Selecting Values Inside a Range

This WHERE phrase selects records in which the UNIT value is between 10,000 and 14,000.

```
WHERE UNITS GE 10000 AND UNITS LE 14000
```

This example is equivalent to:

```
WHERE UNITS GE 10000
WHERE UNITS LE 14000
```

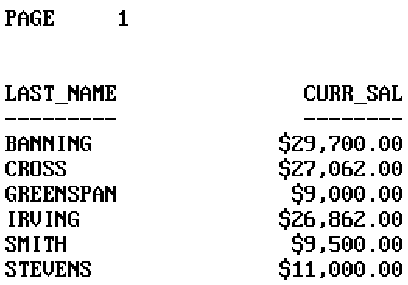
Example

Selecting Values Outside a Range

This request lists employees whose salary is either less than 12,000 or greater than 22,000.

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

The request produces this report:



```
PAGE     1


LAST_NAME              CURR_SAL
---------              --------
BANNING              $29,700.00
CROSS                $27,062.00
GREENSPAN             $9,000.00
IRVING               $26,862.00
SMITH                 $9,500.00
STEVENS              $11,000.00
```

Missing Data Tests

When creating report requests, you may want to test for missing data. Fields that have missing data must have the MISSING attribute set to ON, in the Master File, in order to make this test work. For information on missing data, see Chapter 13, *Handling Records With Missing Field Values*.

Syntax How to Test for Missing Data

The syntax to screen on missing data is:

```
WHERE fieldname {EQ|IS} MISSING
```

These alternatives are synonymous.

Syntax How to Test for Existing Data

To test whether fields are present, the syntax is:

```
WHERE fieldname {NE|IS-NOT} MISSING
```

Character String Screening: CONTAINS and OMITS

The CONTAINS operator tests alphanumeric fields when used with WHERE, and both alphanumeric and text fields when used with IF. If the characters in the given literal or literals appear anywhere within the characters of the field value, the test is passed. For instance, the characters JOHN are contained in JOHNSON and would be selected by the following phrase:

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

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

Note that the field name being tested must appear on the left side of the CONTAINS operator.

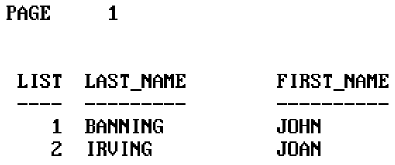
The OMITS operator also tests alphanumeric fields when used with WHERE, and both alphanumeric and text fields when used with IF. It is the opposite of CONTAINS; if the characters of the given literal or literals appear anywhere within the characters of the field's value, the test fails. In the following example, any last name without the string JOHN will be selected:

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

CONTAINS or OMITS tests are useful when the exact spelling of a value is not known. As long as you know that a specific string appears within the value, you can retrieve the desired data. For example:

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

This request will produce all names that contain the letters ING:



```
PAGE     1


 LIST  LAST_NAME        FIRST_NAME
 ----  ---------        ----------
    1  BANNING          JOHN
    2  IRVING           JOAN
```

Screening Masked Fields With LIKE Operators

A mask is an alphanumeric pattern you supply that FOCUS compares to characters in a data field. The data field must have an alphanumeric format (A). You can use the LIKE and NOT LIKE operators to perform screening on masked fields.

To search for records with the LIKE operator, specify the following syntax:

```
WHERE field LIKE 'mask'
```

To reject records based on the mask value, specify either of the following syntax:

```
WHERE field NOT LIKE 'mask'
```

```
WHERE NOT field LIKE 'mask'
```

There are two wildcard characters that you can use in a mask: the underscore (_) indicates that any character in that position is acceptable; the percent sign (%) allows any following sequence of zero or more characters.

Restrictions on Masking Characters

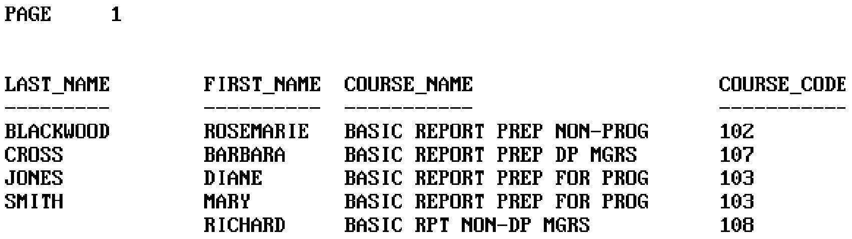
- The wildcard characters dollar sign ($) and dollar sign with an asterisk ($*), which are used with IS operators, are treated as literals with LIKE operators.
- Masking with the characters $ and $* is not supported for compound WHERE clauses that use the AND or OR logical operators.

Example Searching for Records With LIKE

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

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

The request generates this report:



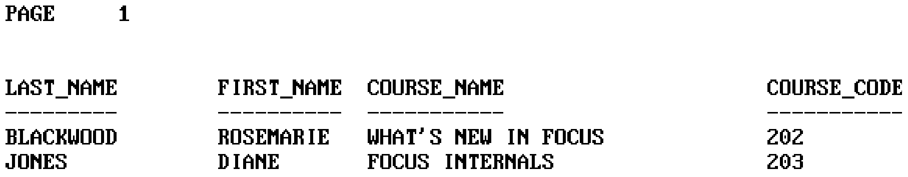
```
PAGE     1


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

Example Searching for Records With a Mask

If you want to see which employees have taken a FOCUS course, but you do not know where the word FOCUS appears in the title, bracket the word FOCUS with wildcards (which is equivalent to using the CONTAINS operator):

```
TABLE FILE EMPLOYEE
PRINT COURSE_NAME COURSE_CODE
BY LAST_NAME BY FIRST_NAME
WHERE COURSE_NAME LIKE '%FOCUS%'
END
```

The request produces this report:

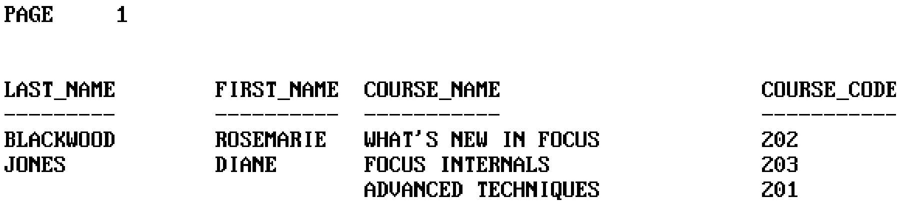


```
PAGE     1


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

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

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

The request produces the following report:



```
PAGE     1


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

ESCAPE Character for LIKE

You can use an escape character in the LIKE syntax to instruct FOCUS not to treat the character ‘%’ or ‘_’ as a special character in a specific request. This technique enables you to search for these characters in the data. 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:

- The percent sign (%) to indicate any number of characters; and
- The underscore (_) to indicate a single character in a specified position.

The ESCAPE character 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 ‘_’.

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 in single quotation marks. When the ESCAPE character is used in the pattern immediately preceding the special characters ‘%’ or ‘_’, FOCUS is instructed to treat the special characters 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 these models, the syntax is:

```
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 Usage Notes for ESCAPE Characters

- The use of an ESCAPE character in front of any character other that ‘%’, ‘_’, and itself will be ignored.
- Only one ESCAPE character can be used per LIKE phrase.
- If a WHERE clause is used with lazy ORs, the ESCAPE must be on the first phrase and will apply to all subsequent phrases in that WHERE clause. For example:

  ```
  WHERE field LIKE 'ABCg_' ESCAPE 'g' OR 'ABCg%' OR 'g%ABC'
  ```

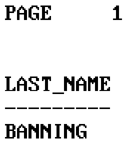
Screening Masked Fields With IS Operators

A mask is an alphanumeric pattern you supply that FOCUS compares to characters in a data field. The data field must have an alphanumeric format (A). You can use the IS and IS-NOT operators to perform screening on masked fields.

The IS and IS-NOT operators may be used for screening on masked fields. The wildcard character is the dollar sign ($). The dollar sign indicates to FOCUS that any character in that position is acceptable. The dollar sign acts as a place holder, as in the following example:

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

The request produces the following report:



```
PAGE     1


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

The selection criteria in the preceding example retrieves only last names that are seven characters long and whose first three characters are BAN. Characters four through seven can be anything, but the remaining characters (8 through 15) must be blank.

To retrieve records with unspecified lengths, use the dollar sign followed by an asterisk ($*). For example,

```
WHERE LAST_NAME IS 'BAN$*'
```

tells FOCUS to search for last names that start with the letters BAN, regardless of the name's length. The characters $* reduce typing, and enable you to define a screen mask without knowing the exact length of the field you wish to retrieve.

Restrictions on Masking Characters

- The wildcard characters dollar sign ($) and dollar sign with an asterisk ($*), which are used with IS operators, are treated as literals with LIKE operators.
- Masking with the characters $ and $* is not supported for compound WHERE clauses that use the AND or OR logical operators.

Qualifying Parent Segments Using INCLUDES and EXCLUDES

INCLUDES and EXCLUDES work only with multi-segment FOCUS files. They test whether instances of a given field in a child segment include or exclude all literals in a list. INCLUDES and EXCLUDES retrieve only parent records. You cannot print or list any field in the same segment as the field specified for the INCLUDES or EXCLUDES test.

Literals containing embedded blanks must be specified with single quotation marks.

To use more than one EXCLUDES or INCLUDES phrase in a request, begin each phrase on a separate line; do not connect them with AND or OR.

Note: If you code WHERE JOBCODE INCLUDES A01 OR B01, you will get both, as if you had used AND.

Example Using INCLUDES and EXCLUDES

Consider the following example:

```
WHERE JOBCODE INCLUDES A01 AND B01
```

For a record to be selected, its JOBCODE field must have values of both A01 and B01. If either one is missing, the record will not be selected for the report.

If the selection criterion is

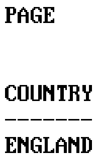
```
WHERE JOBCODE EXCLUDES A01 AND B01
```

every record which does not have both values is selected for the report.

For example, in the CAR database, only England produces Jaguars and Jensens, and so the request

```
TABLE FILE CAR
PRINT COUNTRY
WHERE CAR INCLUDES JAGUAR AND JENSEN
END
```

generates this report:



```
PAGE     1


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

Setting Limits on the Number of Records Read

For some reports, a limited number of records is satisfactory. Therefore, when a stated limit is found, record retrieval can stop. For instance, this is useful when:

- You are designing a new report, and you need only a few records from the actual data file to test your design.
- The database administrator needs to limit the size of reports by placing an upper limit on retrieval from very large data sources. This limit is attached to the user’s password.
- You know the number of records that meet the test criteria. You can specify that number so that the search does not continue beyond the last record that meets the criteria. For example, suppose only ten employees use electronic transfer of funds and you want to retrieve only those records. The record limit would be ten, and retrieval would stop when the tenth record is retrieved. The data source will not be searched any further.

Syntax How to Limit the Number of Records Read

There are two ways to limit the number of records retrieved. You can use the syntax

```
WHERE RECORDLIMIT EQ n
```

where:

n

Is a number greater than 0, and indicates the number of records to be retrieved. This syntax can be used with FOCUS and non-FOCUS data sources.

For non-FOCUS data sources, you can also use the syntax

```
WHERE READLIMIT EQ n
```

where:

n

Is a number greater than 0, and indicates the number of read operations (not records) issued by FOCUS.

This syntax operates on all non-FOCUS data sources.

If an attempt is made to apply the READLIMIT test to a FOCUS database, the request is processed correctly, but the READLIMIT phrase is ignored.

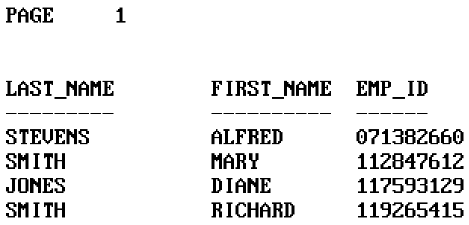
Example Limiting the Number of Records Read

Consider the following example:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME AND EMP_ID
WHERE RECORDLIMIT EQ 4
END
```

Four records are retrieved generating a four-line report.

The request produces the following report:



```
PAGE     1


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

Selecting Records Using IF Phrases

The IF phrase selects records to be included in a report, and offers a subset of the functionality of WHERE.

The syntax for record selection using the IF phrase is

```
IF fieldname operator literal [OR literal]
```

where:

fieldname
Is the field you want to test (the test value).

operator
Is the type of selection operator you want. Valid operators are described in *Types of IF Selection Tests* on page 5-28.

literal
Can be the MISSING keyword (as described in *Types of WHERE Selection Tests* on page 5-10), or alphanumeric or numeric values that are in your data file, with the word OR between each value.

Note: All literals that contain blanks (New York City, for example) and all date literals must be enclosed within single quotation marks.

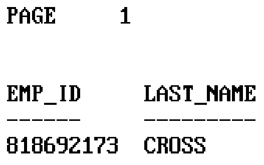
The IF phrase alone cannot be used to create compound expressions from simple expressions joined by AND and OR logical operators. Compound logic requires that the IF phrase be used with the DEFINE command, as described in Chapter 8, *Using Expressions*.

Example Using Multiple IF Phrases

You can use as many IF phrases as necessary to define all your selection criteria, as illustrated in the following example:

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

All of these criteria must be satisfied in order for a record to be included in a report. The request creates this report:



```
PAGE     1


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

Reading Selection Values From a File

Instead of typing all of the selection value literals in the IF phrase, you can store them in a file and refer to the file in the report request. You can then retrieve the data from this file and substitute it as the literals in the selection criterion. This test has the following advantages:

- You can save time by coding a large set of selection values just once, and ensure consistency by maintaining it in just one location, yet use it in as many report requests as you wish.
- If the selection values already exist in a data source, you can quickly create a file of selection values by generating a report and saving the output in an extract file—a HOLD or SAVE file. You can then read selection values from the file.

  If you use a HOLD file, it must either be in internal format (the default), or in character (ALPHA) format; if you use a SAVE file, it must be in character format (the default). You can also use a SAVB file if the selection values are alphanumeric. See Chapter 12, *Saving and Reusing Your Report Output*, for information on HOLD, PCHOLD, SAVE, and SAVB files. Note that in MVS, a HOLD file in internal format that is used for selection values must be allocated to ddname HOLD (the default); the other extract files used for this purpose can be allocated to any ddname.

The syntax for referring to a file of selection values is

```
IF fieldname operator (file) [OR (file) ... ]
```

where:

fieldname
Is any valid field name or alias.

operator
Is the EQ, IS, NE, or IS-NOT operator (see *Types of IF Selection Tests* on page 5-28).

file
Is the name of the file. For MVS, this is the ddname assigned by a FOCUS DYNAM or TSO ALLOCATE command. For CMS, this is the ddname assigned by a FILEDEF command. The total of all files can contain up to 32,767 literals, including newline and other formatting characters.

Only one data value can appear on one line (only the first data value encountered on each line is used). There may be other information in the external file on each line following the value that is actually being tested.

The first data value must start in Column 1 of the external file. The data values are assumed to be in character format. Numeric digits are converted to internal computational numbers where needed (that is, binary integer, and so forth).

Sets of file names may be used, separated by the word OR. Also, actual literals may be mixed with the file names. For example:

```
IF fieldname operator (filename) OR literal...etc...
```

Note:

- When there is more than one IF test, the limit applies to each IF test individually.
- The limit on files used in a WHERE phrase is 32,767 literals per WHERE phrase.

Example Reading Selection Values From a File

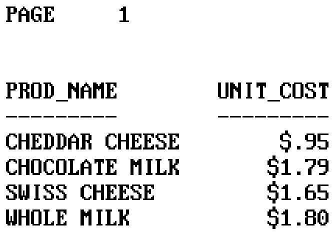
For example, the following request

```
TABLE FILE PROD
SUM UNIT_COST
BY PROD_NAME
IF PROD_CODE IS (EXPER)
END
```

together with the EXPER file containing the following records

```
B10
B12
B17
B19
B20
```

generates this report:



```
PAGE     1


PROD_NAME         UNIT_COST
---------         ---------
CHEDDAR CHEESE         $.95
CHOCOLATE MILK        $1.79
SWISS CHEESE          $1.65
WHOLE MILK            $1.80
```

The value of PROD_CODE is compared to the values in the EXPER file for the selection criterion.

VSAM Record Selection Efficiencies

The most efficient way to retrieve selected records from a VSAM KSDS file is by applying an IF screening test against the primary key. This results in a direct read of the data, that is, through the file's index. Only those records that you request are retrieved from the file. The alternative method of retrieval, the sequential read, forces the interface to bring all the records into storage.

Selection criteria based on the entire primary key or on a subset of the primary key causes direct reads using the index. A partial key is any contiguous part of the primary key beginning with the first byte.

IF selection tests performed against defined fields can take advantage of these efficiencies as well, if the full or partial key is imbedded in the defined field.

The EQ and IS relations realize the greatest performance improvement over sequential reads. When testing on a partial key, they retrieve only the first segment instance of the screening value. To retrieve subsequent instances, NEXT logic is used.

Screening relations GE, FROM, FROM-TO, GT, EXCEEDS, IS-MORE-THAN, and NOT-FROM-TO all obtain some benefit from direct reads. The following example uses the index to find the record containing primary key value 66:

```
IF keyfield GE 66
```

Then, it continues to retrieve records by sequential processing because VSAM stores records in ascending key sequence. FOCUS does not attempt the direct read when the IF screening conditions NE, IS-NOT, CONTAINS, OMITS, LT, IS-LESS-THAN, LE, and NOT-FROM are present in the TABLE request.

Reporting From Files With Alternate Indices

Similar performance improvement is available for ESDS and KSDS files which use alternate indices. An alternate index provides access to records in a key sequenced data set based on a key other than the file's primary key.

All benefits and limitations inherent with screening on the primary or partial key are applicable to screening on the alternate index or partial alternate index. Refer to the previous section for a discussion of these efficiencies.

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

Types of IF Selection Tests

You can use the following operators with the IF phrase:

Operator	**Meaning**
`EQ` `IS`	Tests for and selects values equal to the test value.
`NE` `IS-NOT`	Tests for and selects values not equal to the test value.

`GE`
`FROM`
`IS-FROM`
	Tests for and selects values greater than or equal to the test value (based on 0 to 9 for numeric values, A to Z and a to z for alphanumeric values).

`GT`
`EXCEEDS`
`IS-MORE-THAN`
	Tests for and selects values greater than the test value.

`LT` `IS-LESS-THAN`	Tests for and selects values less than the test value.
`LE` `TO`	Tests for and selects values less than or equal to the test value.
`IS-FROM` *lower* `TO` *upper*	Tests for and selects values within a range of values.
`NOT-FROM` *lower* `TO` *upper*	Tests for and selects values that are not within a range of values.

`IS MISSING`
`IS-NOT MISSING`
`NE MISSING`
	Tests whether a test value is missing or not.

`CONTAINS`	Tests for and selects values that include a character string matching test value. The string can occur in any position in the value being tested. Valid with alphanumeric fields.
`OMITS`	Tests for and selects values that do not include a character string matching the test value. The string cannot occur in any position in value being tested. Valid with alphanumeric fields.
`INCLUDES`	Tests whether a chain of values for a given field in a child segment includes all of a list of literals.
`EXCLUDES`	Tests whether a chain of values for a given field in a child segment excludes all of a list of literals.

Note: When you use compound logic in an expression (AND or OR) to select records, you must either use the DEFINE command in conjunction with the IF phrase (described in Chapter 8, *Using Expressions*), or the WHERE phrase (described in *Selections Based on Individual Values: WHERE* on page 5-4).

Assigning Screening Conditions to a File

You can use a filtering mechanism for the TABLE facility to assign screening conditions to a data source. This technique enables you to declare a set of screening conditions, and assign it to a specific data source, 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	**With Filters**
`TABLE FILE CAR…`	`FILTER FILE CAR…`
`WHERE…SEATS GT 5`	`SET FILTER= WHERE SEATS GT 5…`
`TABLE FILE CAR…`	`TABLE FILE CAR…`
`WHERE SEATS GT 5…`	`TABLE FILE CAR…`
`TABLE FILE CAR…`	`TABLE FILE CAR…`
`WHERE SEATS GT 5`	`TABLE FILE CAR…`

Whenever a TABLE request is issued against a data source, all filters that have been activated for that data source 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 condition that is automatically added to every TABLE request against the specified data source as if the IF or WHERE condition were actually coded in the request. All IF/WHERE syntax that is valid in a TABLE request 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. You can declare one or more filters for a data source.

Just declaring a filter for a data source 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 data source.
- 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.

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 the Master File to which the filters will apply.

filter-defines

Are virtual fields declared for use in filters. For more information, see *Filter Defines* on page 5-31.

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 eight characters in length and must be unique for a particular file name.

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 data source fields and defines in the Master File. May not refer to other filter names.

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.
- Do not count toward the 256 verb object limit of TABLE, unlike DEFINEs from DEFINE FILE commands 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 creates the filter UK with a WHERE condition. It also adds a definition for MARK_UP to the CAR file’s set of filter-defines. 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 request.

```
FILTER FILE CAR ADD
MARK_UP/D7=RCOST-DCOST;
NAME=UK
WHERE MARKUP GT 1000
END
```

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.

Note: The maximum number of filters set ON for a data source 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 request.

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.

Example Activating Filters

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).

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 Master File.

`*`
: Displays filters for all Master 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
```

If no filters are defined, the following message displays

```
NO FILTERS DEFINED
```

or, if filters are defined, the following screen displays:

```
Set File     Filter name Description
--- -------- ----------- -----------------------------------
    CAR      ROB         Rob's selections
*   CAR      PETER       Peter’s selections for CAR
*   EMPLOYEE DAVE        Dave’s tests
    EMPLOYEE BRAD        Brad’s tests
```

To query filters for the CAR database, issue:

```
? FILTER CAR
```

If no filters are defined for the CAR file, the following message displays

```
NO FILTERS DEFINED FOR FILE NAMED CAR
```

or, if filters are defined for the CAR file, the following screen displays:

```
Set File     Filter name Description
--- -------- ----------- -----------------------------------
    CAR      ROB         Rob's selections
*   CAR      PETER       Peter’s selections for CAR
```

To see all active filters, issue the following command

```
? FILTER * SET
Set File     Filter name Description
--- -------- ----------- -----------------------------------
*   CAR      PETER       Peter’s selections for CAR
*   EMPLOYEE DAVE        Dave’s tests
```

The asterisk in the first column indicates that a filter is ON.

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 files remain alive as originally declared.

```
-*****************************
-* 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
```

6 Creating Temporary Fields

Topics:

- The Difference Between COMPUTE and DEFINE
- Defining a Temporary Field: DEFINE
- Computing Temporary Fields: COMPUTE
- Using Subroutines With Temporary Fields

Frequently, a report requires information that does not physically exist, but can be produced from existing data source information. The FOCUS DEFINE and COMPUTE commands enable you to create a variety of new temporary fields.

The limit for the number of DEFINEs allowed in FOCUS 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 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.

Temporary fields, created by the COMPUTE command, are counted against the standard maximum of 256 display fields. Their command syntax accepts the same types of valid expressions. Temporary fields can be used with FOCUS report requests, as well as created for other applications like MODIFY procedures and Dialogue Manager stored procedures (see the *Developing Applications* manual).

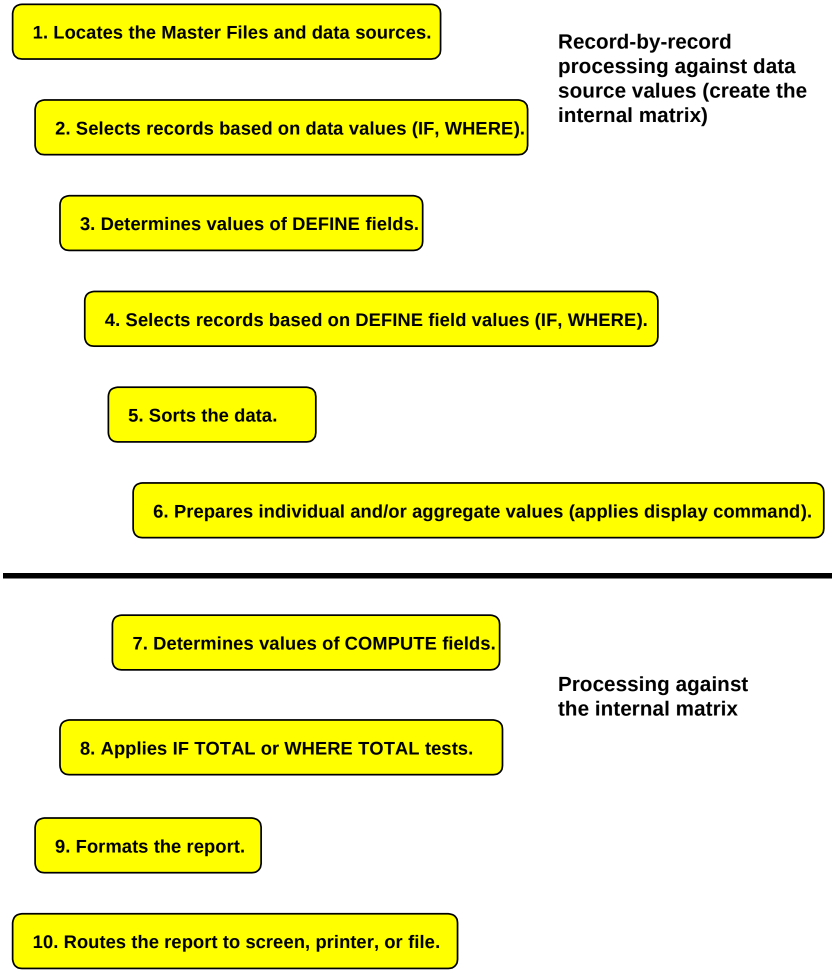
The Difference Between COMPUTE and DEFINE

The major difference between COMPUTE and DEFINE is the point of execution.

- A defined field is just like a real field in the Master File, and the calculation is performed on each retrieved record that passes any screening conditions on real fields. The result of the expression is treated as though it is a real field stored in the data source.
- A computed field, on the other hand, works on the results of a SUM, PRINT, or COUNT command. The COMPUTE field is calculated on the results after all records have been selected, sorted, and summed. The COMPUTE calculation, therefore, is performed using the summed field values of the fields that it references.

The following diagram illustrates how FOCUS processes a report:

How FOCUS Processes a Report



1. Locates the Master Files and data sources.

2. Selects records based on data values (IF, WHERE).

3. Determines values of DEFINE fields.

4. Selects records based on DEFINE field values (IF, WHERE).

5. Sorts the data.

6. Prepares individual and/or aggregate values (applies display command).

Record-by-record processing against data source values (create the internal matrix)

7. Determines values of COMPUTE fields.

8. Applies IF TOTAL or WHERE TOTAL tests.

9. Formats the report.

10. Routes the report to screen, printer, or file.

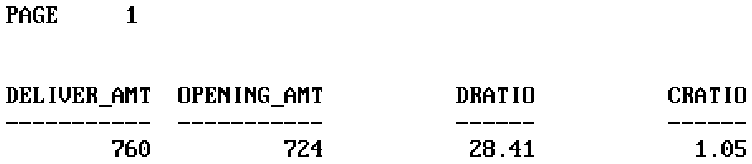
Processing against the internal matrix

Example Distinguishing Between DEFINE and COMPUTE

The difference between DEFINE and COMPUTE is best illustrated when using a SUM command in the request:

```
DEFINE FILE SALES
DRATIO = DELIVER_AMT/OPENING_AMT;
END

TABLE FILE SALES
SUM DELIVER_AMT AND OPENING_AMT AND DRATIO
COMPUTE CRATIO = DELIVER_AMT/OPENING_AMT;
END
```

Both the defined field (DRATIO) and the computed field (CRATIO) are equal to the same expression, but their results are very different:



```
PAGE     1


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

The computed field CRATIO is calculated on the results after all records have been selected, sorted, and summed. The calculation is performed using the summed field values of the fields that it references. The defined field DRATIO is calculated on each retrieved record.

Defining a Temporary Field: DEFINE

DEFINE commands create one or more temporary fields that may then be specified in the report request, as though they were real data source fields. Defined fields remain active throughout the FOCUS session, unless released by a DEFINE CLEAR option, by a later DEFINE request—without the ADD phrase—against the same data file, or by the JOIN command.

Defined fields may also be described in the Master File. These fields are available whenever the data file is used for reporting. Whether defined fields in the Master File can refer to fields located in different segments depends on the FIELDNAME setting; fields created with the DEFINE command can refer to such fields. Also, unlike fields created with the DEFINE command, defined fields in the Master File are not affected by JOIN or DEFINE CLEAR commands.

A defined field is just like a real field in the Master File, and the calculation is performed on each retrieved record that passes any screening conditions on real fields. The result of the expression is treated as though it were a real field stored in the data source.

Syntax How to Define a Temporary Field

You specify the DEFINE command before you begin a TABLE request. It has the following syntax

```
DEFINE FILE filename[.view_fieldname] [CLEAR|ADD]

fieldname[/format]=expression;
fieldname[/format][WITH realfield]=expression;
fieldname[/format] REDEFINES qualifier.fieldname=expression;
.
.
.
END
```

where:

filename

Is the name of the file for which you are defining the new field. You can omit the file name if you have established a current file with the SET FILE command (see the *Developing Applications* manual). Use *filename* if the corresponding TABLE request uses this format. Alternate view TABLE files may require alternate view DEFINE for DEFINEd fields. For an alternate view, you cannot use fields that exceed the 12-character limit or qualified fields in a DEFINE.

view_fieldname

Is the field on which an alternate view is taken in the corresponding report request. You must include this field if the corresponding report request uses this format.

CLEAR

Optional (default). Clears previously defined temporary fields associated with the specified file.

ADD

Optional. Enables you to specify additional temporary fields without releasing any existing temporary fields. Omitting ADD produces the same results as the CLEAR option.

fieldname

Is a name of up to 66 characters, except for text and indexed fields in FOCUS Master Files. Indexed and text field names must be less than or equal to 12 characters long. It can be the name of a new temporary field that you are defining or an existing field declared in the Master File that you want to redefine.

The name can include any combination of letters, digits, and underscores (_) and should begin with a letter.

It is recommended that you do not use field names of the type C*n*, E*n*, and X*n* (where *n* is any sequence of one or two digits) because FOCUS uses these to refer to report columns, HOLD file fields, and other special objects.

format

Is the format of the field. All formats except text fields (TX) are allowed. The default is D12.2. For information on field formats, see the *Describing Data* manual.

`WITH` *`realfield`*

Associates a temporary field with a data source segment containing a real field.

`REDEFINES` *`qualifier.fieldname`*

In cases where the same field name exists in more than one segment, and that field must be redefined or recomputed, the REDEFINES phrase must be used.

`expression`

Can be an arithmetic and/or logical expression or function to establish the value of *fieldname* (see Chapter 8, *Using Expressions*). Each expression must end with a semicolon except the last one, where the semicolon is optional.

`END`

Is required to end the DEFINE file.

Reference Usage Notes for Defining Temporary Fields

- Fields in the expression can be real data fields, data fields in files that are cross-referenced or joined, or previously defined fields. However, when a JOIN is issued for a file, all pre-existing defined fields for that file are cleared except ones defined in the Master File.
- If you wish to join structures using a field defined by a DEFINE command, the DEFINE must follow the JOIN command. See Chapter 14, *Joining Data Files*, for an explanation of reporting on joined files and complete information on the JOIN command.
- Temporary fields declared with the DEFINE attribute in a Master File are not affected by DEFINE CLEAR commands.
- If no field in the expression is in the Master File or has been defined, you need to use the WITH phrase to identify the logical home of the defined calculation. See *Defining Fields Using the WITH Phrase* on page 6-8.
- You may define fields simultaneously (in addition to fields defined in the Master File) for as many files as desired, but only 256 fields may be defined for a single file at one time. The total length of all defined fields and real fields cannot exceed 16,000 characters. This limit may be increased at installation time. Please refer to your FOCUS installation guide.
- When you specify defined fields in a request, they count toward the limit of 256 display fields; otherwise, they are not counted.
- DEFINE fields are only available when the data file is used for reporting. DEFINEs cannot be used with MODIFY.
- A DEFINE may not contain qualified field names on the left-hand side of the expression. In cases where the same field name exists in more than one segment, and that field must be redefined or recomputed, the REDEFINES phrase must be used.

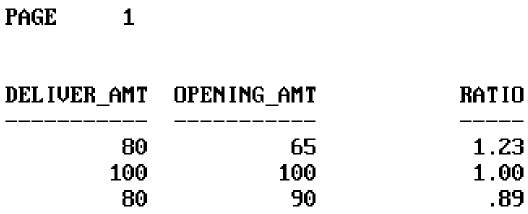
- When the value of FIELDNAME is changed in a FOCUS session, DEFINEs are affected as follows:
  - When the value changes from OLD to NEW or OLD to NOTRUNC, all DEFINEs are cleared.
  - When the value changes from NEW to OLD, all DEFINEs are cleared.
  - When the value changes from NOTRUNC to OLD, all DEFINEs are cleared.
- Using a self-referencing DEFINE such as x=x+1 disables AUTOPATH (see the *Developing Applications* manual).
- Packed fields should not be used in computations with floating point fields.

Example Defining a Temporary Field

In this example, the value of RATIO is calculated by dividing the value of DELIVER_AMT by OPENING_AMT. The DEFINE command enables you to make RATIO a temporary field, which is used in the report request as though it were a real field in the database:

```
DEFINE FILE SALES
RATIO = DELIVER_AMT/OPENING_AMT;
END

TABLE FILE SALES
PRINT DELIVER_AMT AND OPENING_AMT AND RATIO
WHERE DELIVER_AMT GT 50
END
```

This request produces the following report:



```
PAGE     1


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

Defining Multiple Temporary Fields

In some cases, you may wish to have more than one DEFINE command referring to the same file and to use some or all of the defined fields in the report request. The ADD option enables you to specify additional temporary fields without releasing existing ones. If you omit the ADD option, previously defined fields in that file are released.

If you want to clear or release DEFINEs for a particular file, use the CLEAR option.

ADD and CLEAR options are especially useful when you are executing report requests from stored procedures or are creating interactive stored procedures (explained in the *Developing Applications* manual). To see which temporary fields are available, use the ? DEFINE command described in the *Developing Applications* manual.

Example Adding and Clearing Defined Fields

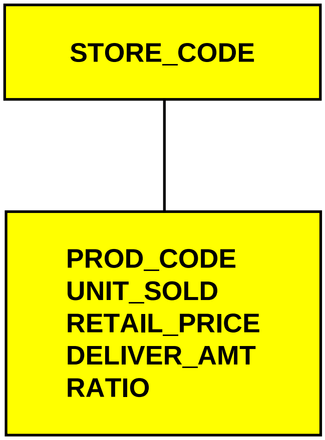
ADD and CLEAR options are both specified in this annotated example:

```
1. DEFINE FILE CAR
   ETYPE/A2=DECODE STANDARD (OHV O OHC O ELSE L);
   END

2. DEFINE FILE CAR ADD
   TAX/D8.2=IF MPG LT 15 THEN .06*RCOST
       ELSE .04*RCOST;
   FCOST = RCOST+TAX;
   END

3. DEFINE FILE CAR CLEAR
   COST = RCOST-DCOST;
   END
```

1. The first DEFINE command creates the ETYPE temporary field for the CAR file. For information about the DECODE function, see Chapter 9, *Using Functions and Subroutines*.
2. Two more temporary fields, TAX and FCOST, are created for the CAR file. The ADD option allows you to reference ETYPE, TAX, and FCOST in future report requests.
3. The CLEAR option clears the three previously defined temporary fields and only the COST temporary field in the last DEFINE is available for further requests.

Defining Fields Using the WITH Phrase

Defined fields have a logical location in the file structure just like permanent data source fields. The logical home of a defined field is on the lowest record segment that has to be accessed in order to evaluate the expression defining the field. The logical home of a defined field determines its time of execution. Consider the following file structure and the following DEFINE command:



```
RATIO = DELIVER_AMT/RETAIL_PRICE ;
```

The expression for RATIO includes at least one real data source field. As far as report capabilities are concerned, the field RATIO is just like a real field in the Master File and is located in the lowest segment.

In some applications, you can have a defined field equal to an expression that contains no real data source fields. Such an expression might refer to only temporary fields or literals. For example,

```
NCOUNT = NCOUNT+1;
```

or

```
DATE = 19990101;
```

Since neither expression contains a data source field (NCOUNT and the literal do not exist in the Master File), FOCUS cannot determine their logical position in the data source. Thus you have to tell FOCUS in which segment you want them to be placed. To associate a defined field with a data source field in a specific segment, use the WITH phrase. The field name following WITH may be any real field in the Master File.

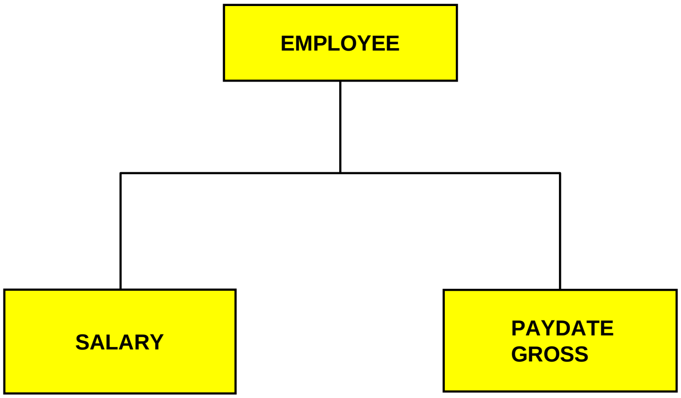
For example, FOCUS places the field NCOUNT in the same segment as UNIT_SOLD. NCOUNT is calculated each time a new segment instance is retrieved:

```
DEFINE FILE PROD
NCOUNT/I5 WITH UNIT_SOLD = NCOUNT+1;
END
```

You may be able to increase the retrieval speed with an external index on the defined field. In this case, you can associate the index with a target segment outside of the segment containing the defined field.

Defining Fields Using Multi-Path Files

The expression of a defined field may include fields from all segments of a file, but they must lie in a unique top-to-bottom path. Different defined fields may, of course, lie along different paths. For example, consider the following file structure:



This file structure would not permit you to write the following expression:

```
NEWAMT = SALARY+GROSS;
```

The expression is invalid because the structure implies that there can be several SALARY segments for a given EMPLOYEE and it is not clear which SALARY to associate with which GROSS.

To accomplish such an operation, you can use the alternate view option explained in Chapter 16, *Improving Data Retrieval*.

Increasing the Speed of DEFINE Calculations

Calculations for DEFINE are fully compiled, by default, into machine code when the request is parsed. The machine code is then executed to perform the calculations at run time. This increases the speed and efficiency of your calculations.

Certain calculations are ineligible for compilation:

- Calculations that involve any function (for example, user functions), except for EDIT, DECODE, and LAST.
- Calculations that test for existing data (IF field IS-NOT MISSING) or that result in a missing field (TEMP/A4 MISSING ON= ...).
- Calculations that involve fields with date formats. (See the table of date formats in the FORMAT attribute description in the *Describing Data* manual.)
- Calculations that use exponentiation (10**2).

Expressions used for calculations are described in the following subsections.

In earlier FOCUS releases (prior to Release 5.5), calculations for DEFINE were converted to an internal form and interpreted at execution time. The ineligible calculations listed above are automatically processed using this conversion logic.

To prevent compilation into machine code, enter the following SET option before issuing a DEFINE command:

```
SET COMPUTE = OLD
```

Calculations are processed by the conversion logic until the session ends or the SET option is changed back to NEW. OLD requires less memory but takes longer to execute. See the *Developing Applications* manual for information about SET COMPUTE.

Computing Temporary Fields: COMPUTE

The COMPUTE command creates one or more temporary fields in the report request. Computed fields do not remain active for the duration of the FOCUS session; they are available for only the specified report request.

A computed field works on the results of a SUM, PRINT, or COUNT command. The COMPUTE field is calculated on the results after all records have been selected, sorted, and summed. The COMPUTE calculation, therefore, is performed using the summed field values of the fields that it references.

Syntax How to Compute Temporary Fields

You specify the COMPUTE command in the body of the report request. You can compute more than one field with a single COMPUTE command.

The syntax is

```
TABLE FILE filename
display_command...
[AND] COMPUTE
fieldname [/format] = expression;
fieldname [/format] = expression; [NOPRINT|AS 'title'] [IN [+]n]
.
.
.
END
```

where:

display_command

Is a display command (PRINT, LIST, SUM, or COUNT). It must occur before the keyword COMPUTE.

fieldname

Is the name of the computed field.

The name can be up to 66 characters long and can include any combination of letters, digits, and underscores (_), and should begin with a letter. Other characters are not recommended and may cause problems in some operating environments or when resolving expressions.

It is recommended that you do not use field names of the type C*n*, E*n*, and X*n* (where *n* is any sequence of one or two digits), because FOCUS uses these to refer to report columns, HOLD file fields, and other special objects.

format

Is the format of the field. All formats except text fields (TX) are allowed. The default is D12.2. For information on formats, see the *Describing Data* manual.

`expression`

Can be an arithmetic and/or logical expression or function (see Chapter 8, *Using Expressions*). Each field used in the expression must be part of the request. Each expression must end with a semicolon.

`NOPRINT`

Suppresses the printing of the field. See Chapter 10, *Customizing Tabular Reports*, for more information.

`AS '`*`title`*`'`

Changes the name of the computed field. See Chapter 10, *Customizing Tabular Reports,* for more information.

`IN`

Specifies the location of the column. See Chapter 10, *Customizing Tabular Reports*, for more information.

Reference Usage Notes for Computed Fields

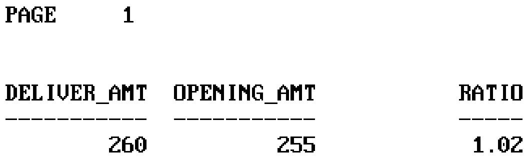
- If you specify any optional COMPUTE phrases, and you compute additional fields following the phrases, you must include the keywords COMPUTE or AND COMPUTE before specifying the additional fields.
- You can rename and justify column totals and row totals. See the examples in Chapter 7, *Including Totals and Subtotals in Your Report.*
- Expressions in a COMPUTE command can include fields with prefix operators (as described earlier in Chapter 2, *Displaying Report Data).* For more information on valid expressions, see Chapter 8, *Using Expressions.*
- Packed fields should not be used in computations with floating point fields.
- Fields referred to in a COMPUTE expression will be counted in the 256-field limit, unless they have been previously referenced.
- While not required, it may sometimes be desirable to put the BY and/or ACROSS phrases before the display command to reduce the number of implied display fields. WHERE and IF phrases can be placed before or after the COMPUTE command.

Example Computing a Temporary Field

For instance, in this request, the COMPUTE command creates a temporary field based on summarized field values. RATIO is computed using DELIVER_AMT and OPENING_AMT for all delivery amounts greater than 50. RATIO is only available for this report request.

```
TABLE FILE SALES
SUM DELIVER_AMT AND OPENING_AMT AND
COMPUTE RATIO = DELIVER_AMT/OPENING_AMT;
WHERE DELIVER_AMT GT 50
END
```

This request produces the following report:



```
PAGE     1


DELIVER_AMT  OPENING_AMT           RATIO
-----------  -----------           -----
        260          255            1.02
```

Using Positional Column Referencing With Computed Fields

In a COMPUTE command, it is sometimes convenient to refer to a field by its report column position rather than its name. This option is especially useful when the same field is specified for several report columns.

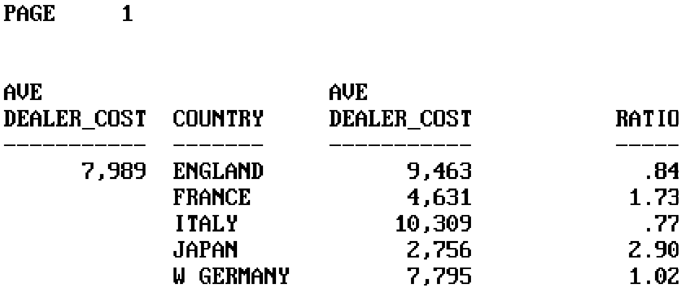
Column referencing becomes essential when using the same field name in a variety of ways. The columns produced by display commands can be referred to as C1 for the first column, C2 for the second column, and so forth. The BY field columns are not counted.

Example Using Positional Column Referencing

The following example demonstrates positional field references in a COMPUTE:

```
TABLE FILE CAR
SUM AVE.DEALER_COST
SUM AVE.DEALER_COST AND COMPUTE RATIO=C1/C2;
BY COUNTRY
END
```

The request produces this report. The columns produced by display commands can be referred to as C1 for the first column (AVE.DEALER_COST), C2 for the second column (AVE.DEALER_COST BY COUNTRY), and so forth. The BY field columns are not counted.



```
PAGE      1


AVE                         AVE
DEALER_COST  COUNTRY        DEALER_COST          RATIO
-----------  -------        -----------          -----
      7,989  ENGLAND              9,463            .84
             FRANCE               4,631           1.73
             ITALY               10,309            .77
             JAPAN                2,756           2.90
             W GERMANY            7,795           1.02
```

Using COMPUTE and ACROSS

If the COMPUTE phrase is part of a display command, a new column is calculated for each set of field values.

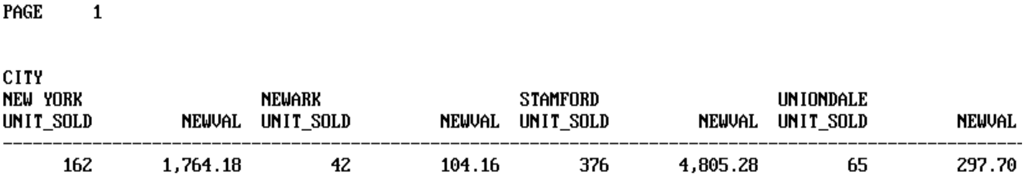
If the COMPUTE phrase is issued right after an ACROSS phrase, only a recap type of calculation is performed once for all columns.

Example Using Compute as Part of a Display Command

Note the following request example:

```
TABLE FILE SALES
SUM UNIT_SOLD COMPUTE NEWVAL = UNIT_SOLD * RETAIL_PRICE;
ACROSS CITY
END
```

The first page of the report that results is shown below:



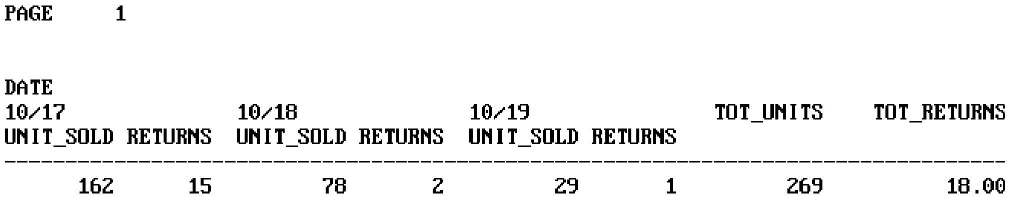
```
PAGE     1


CITY
NEW YORK                 NEWARK                   STAMFORD                 UNIONDALE
UNIT_SOLD        NEWVAL  UNIT_SOLD        NEWVAL  UNIT_SOLD        NEWVAL  UNIT_SOLD        NEWVAL
------------------------------------------------------------------------------------------------------
      162      1,764.18         42        104.16        376      4,805.28         65        297.70
```

Example Using Compute After an Across Phrase

For example:

```
TABLE FILE SALES
SUM UNIT_SOLD AND RETURNS
WHERE DATE GE '010' AND DATE LE '1031'
ACROSS DATE
COMPUTE
TOT_UNITS/D5=C1 + C3 + C5;
TOT_RETURNS = C2 + C4 + C6;
END
```

The request generates the following report. C1, C2, C3, C4, C5, and C6 are positional column references.



```
PAGE     1


DATE
10/17                10/18                10/19                  TOT_UNITS    TOT_RETURNS
UNIT_SOLD RETURNS    UNIT_SOLD RETURNS    UNIT_SOLD RETURNS
------------------------------------------------------------------------------------------
      162       15          78        2          29        1           269          18.00
```

Sorting Computed Fields

You cannot sort by a computed field, although you can sort by a defined field. If you want to sort by a computed field, you must create a HOLD file that contains the results of the request and write another request that uses the HOLD file as the data file. For more information on HOLD files, see Chapter 12, *Saving and Reusing Your Report Output.*

Screening on Computed Fields

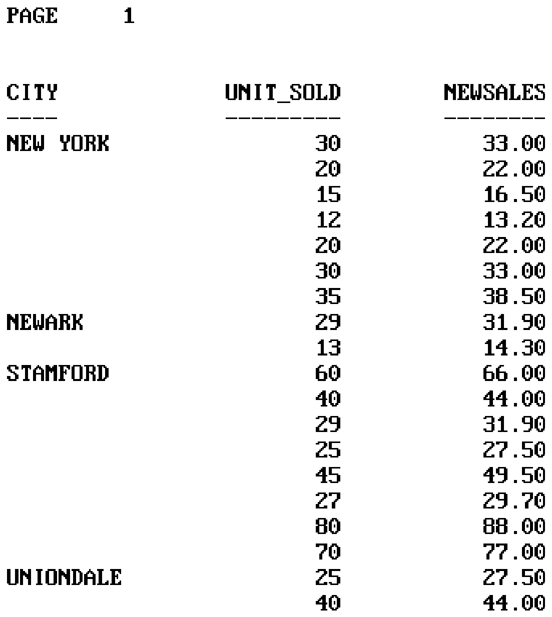
You can screen on values produced by COMPUTE commands by using the WHERE TOTAL test, as described in Chapter 5, *Selecting Records for Your Report*.

Example Screening on Computed Fields

Note the following report request.

```
TABLE FILE SALES
PRINT UNIT_SOLD
AND COMPUTE NEWSALES = UNIT_SOLD * 1.1;
BY CITY
WHERE TOTAL NEWSALES GT 10
END
```

The request produces this report:



```
PAGE     1


CITY              UNIT_SOLD         NEWSALES
----              ---------         --------
NEW YORK                 30            33.00
                         20            22.00
                         15            16.50
                         12            13.20
                         20            22.00
                         30            33.00
                         35            38.50
NEWARK                   29            31.90
                         13            14.30
STAMFORD                 60            66.00
                         40            44.00
                         29            31.90
                         25            27.50
                         45            49.50
                         27            29.70
                         80            88.00
                         70            77.00
UNIONDALE                25            27.50
                         40            44.00
```

Using Subroutines With Temporary Fields

Any subroutine name encountered in a DEFINE or COMPUTE expression that is not recognized as a FOCUS-supplied name is assumed to be a user-written subroutine. These subroutines are loaded dynamically when needed. They are coded by users and reside in a library that is available at the time they are referenced. See Chapter 9, *Using Functions and Subroutines*, for more information.

7 Including Totals and Subtotals in Your Report

Topics:

- Calculating Row and Column Totals
- Adding Section Totals and Grand Totals to Your Reports
- SUB-TOTAL and SUBTOTAL
- SUMMARIZE and RECOMPUTE
- RECAP and COMPUTE
- Suppressing Grand Totals: NOTOTAL
- Conditionally Displaying Summary Lines and Text: WHEN

To help you and others interpret the detailed information presented in a report, FOCUS enables you to summarize numeric information using grand totals and subtotals. These summary lines may be used to clarify or highlight information in numeric or matrix reports.

Calculating Row and Column Totals

To produce totals for rows or columns of numbers and to make a report easy to read and/or interpret, use the FOCUS keywords ROW-TOTAL and COLUMN-TOTAL, respectively.

- ROW-TOTAL produces a new column containing the sum of all numbers in each row.
- COLUMN-TOTAL produces a final row on the report which contains the totals for each column of numbers.

Syntax How to Calculate Row and Column Totals

The syntax is

```
display_command fieldname  AND ROW-TOTAL  [alignment] [/format] [AS 'name']
display_command fieldname  AND COLUMN-TOTAL [alignment]  [AS 'name']
```

where:

alignment

Specifies the alignment of the ROW-TOTAL or COLUMN-TOTAL label.

`/R` right justifies the label.

`/L` left justifies the label.

`/C` centers the label.

format

Reformats the ROW-TOTAL.

name

Is the new name for the ROW-TOTAL or COLUMN-TOTAL label.

You may also specify row or column totals with the ON TABLE phrase. Field names are optional with COLUMN-TOTAL and cannot be listed with ROW-TOTAL. Use the following syntax:

```
ON TABLE  ROW-TOTAL [alignment] [/format] [AS 'name'] [fieldname fieldname fieldname]
ON TABLE  COLUMN-TOTAL [alignment] [AS 'name'] [fieldname fieldname fieldname]
```

Reference Usage Notes for Row and Column Totals

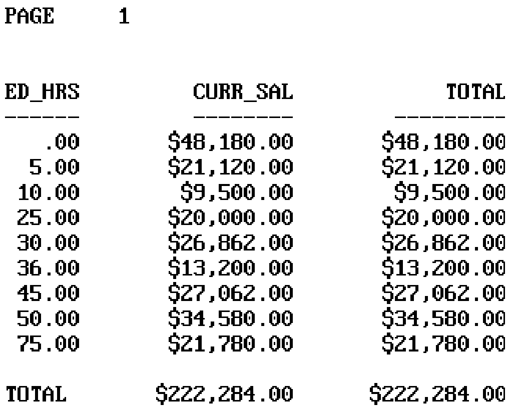
- If one field is summed, the format of the row total is the same as the format of the field. For instance, if the format of CURR_SAL is D12.2M, the format of the row total is also D12.2M.
- When you are summing fields with different formats, a default format of D12.2 is used for the total.

Example Calculating Row and Column Totals

For example:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AND ROW-TOTAL AND COLUMN-TOTAL
BY ED_HRS
END
```

This request produces the following report:



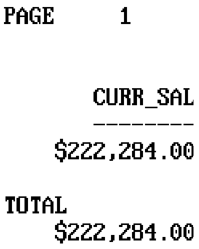
```
PAGE     1


ED_HRS         CURR_SAL            TOTAL
------         --------        ---------
   .00       $48,180.00       $48,180.00
  5.00       $21,120.00       $21,120.00
 10.00        $9,500.00        $9,500.00
 25.00       $20,000.00       $20,000.00
 30.00       $26,862.00       $26,862.00
 36.00       $13,200.00       $13,200.00
 45.00       $27,062.00       $27,062.00
 50.00       $34,580.00       $34,580.00
 75.00       $21,780.00       $21,780.00

TOTAL       $222,284.00      $222,284.00
```

Example Using Column Totals With ON TABLE

For example:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL
ON TABLE COLUMN-TOTAL
END
```

This request produces the following report:



```
PAGE     1


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

TOTAL
    $222,284.00
```

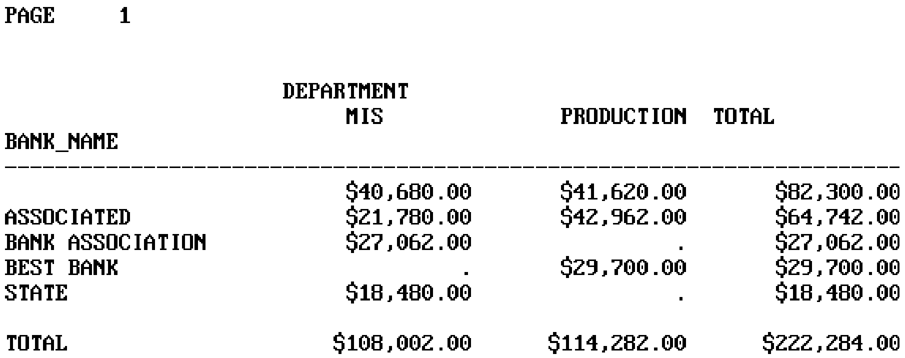
Example Using Row and Column Totals With Matrix Reports

ROW-TOTAL and COLUMN-TOTAL are useful in matrix reports (for example, when using both ACROSS and BY phrases together).

For example:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AND ROW-TOTAL AND COLUMN-TOTAL
BY BANK_NAME
ACROSS DEPARTMENT
END
```

This request produces the following report:

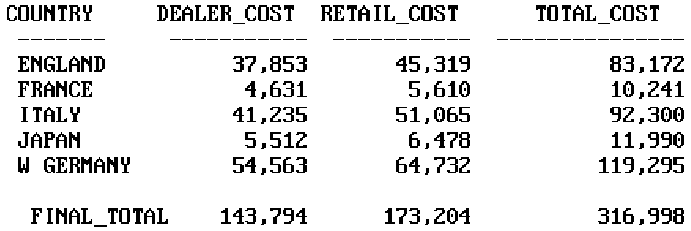


```
PAGE     1


                     DEPARTMENT
                          MIS              PRODUCTION  TOTAL
BANK_NAME
-----------------------------------------------------------------------
                          $40,680.00       $41,620.00       $82,300.00
ASSOCIATED                $21,780.00       $42,962.00       $64,742.00
BANK ASSOCIATION          $27,062.00                .       $27,062.00
BEST BANK                          .       $29,700.00       $29,700.00
STATE                     $18,480.00                .       $18,480.00

TOTAL                    $108,002.00      $114,282.00      $222,284.00
```

Example Renaming Row and Column Totals

The next request renames and centers the ROW-TOTAL and COLUMN-TOTAL:

```
TABLE FILE CAR
SUM DCOST
RCOST ROW-TOTAL/C/D12 AS 'TOTAL_COST'
BY COUNTRY
ON TABLE COLUMN-TOTAL/C AS 'FINAL_TOTAL'
END
```

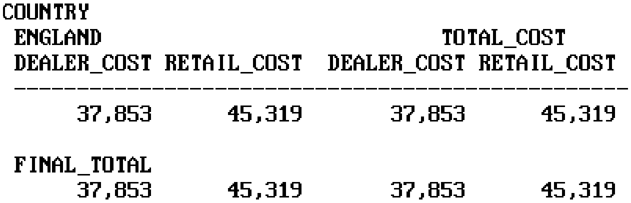


```
COUNTRY     DEALER_COST  RETAIL_COST      TOTAL_COST
 -------     -----------  -----------  ---------------
 ENGLAND          37,853       45,319           83,172
 FRANCE            4,631        5,610           10,241
 ITALY            41,235       51,065           92,300
 JAPAN             5,512        6,478           11,990
 W GERMANY        54,563       64,732          119,295

  FINAL_TOTAL    143,794      173,204          316,998
```

Example Using Row and Column Totals With ACROSS

This request renames and centers the ROW-TOTAL and COLUMN-TOTAL with ACROSS:

```
TABLE FILE CAR
SUM DCOST
RCOST ROW-TOTAL/C/D12 AS 'TOTAL_COST'
ACROSS COUNTRY
IF COUNTRY EQ 'ENGLAND'
ON TABLE COLUMN-TOTAL/C AS 'FINAL_TOTAL'
END
```



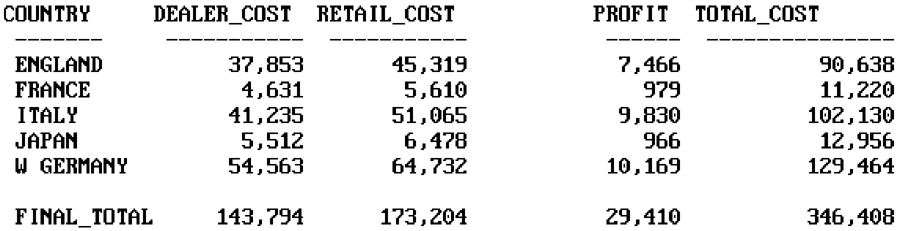
```
COUNTRY
 ENGLAND                           TOTAL_COST
 DEALER_COST RETAIL_COST  DEALER_COST RETAIL_COST
 -------------------------------------------------
      37,853      45,319       37,853      45,319

 FINAL_TOTAL
      37,853      45,319       37,853      45,319
```

Note: Reformatting the ROW-TOTAL and COLUMN-TOTAL field will only be successful with single verb requests. Multi-verb requests will not reformat the row and column total values.

Example Using Row and Column Totals With COMPUTEs

This example renames and left justifies the ROW-TOTAL and COLUMN-TOTAL with COMPUTE:

```
TABLE FILE CAR
SUM DCOST RCOST
COMPUTE PROFIT/D12=RCOST-DCOST;
ROW-TOTAL/L/D12 AS 'TOTAL_COST'
BY COUNTRY
ON TABLE COLUMN-TOTAL/L AS 'FINAL_TOTAL'
END
```



```
COUNTRY     DEALER_COST  RETAIL_COST          PROFIT  TOTAL_COST
 -------     -----------  -----------           ------  ---------------
 ENGLAND          37,853       45,319            7,466           90,638
 FRANCE            4,631        5,610              979           11,220
 ITALY            41,235       51,065            9,830          102,130
 JAPAN             5,512        6,478              966           12,956
 W GERMANY        54,563       64,732           10,169          129,464

 FINAL_TOTAL     143,794      173,204           29,410          346,408
```

Adding Section Totals and Grand Totals to Your Reports

Frequently, reports contain detailed information that is broken down into subsections for which simple column and row totals do not provide adequate summaries. In these instances, it is more useful to look at subtotals for particular sections and a grand total at the end of the report.

FOCUS provides six commands that you can add to your report requests to create section subtotals and grand totals:

- SUB-TOTAL and SUBTOTAL
- SUMMARIZE and RECOMPUTE (Use with computed fields.)
- RECAP and COMPUTE

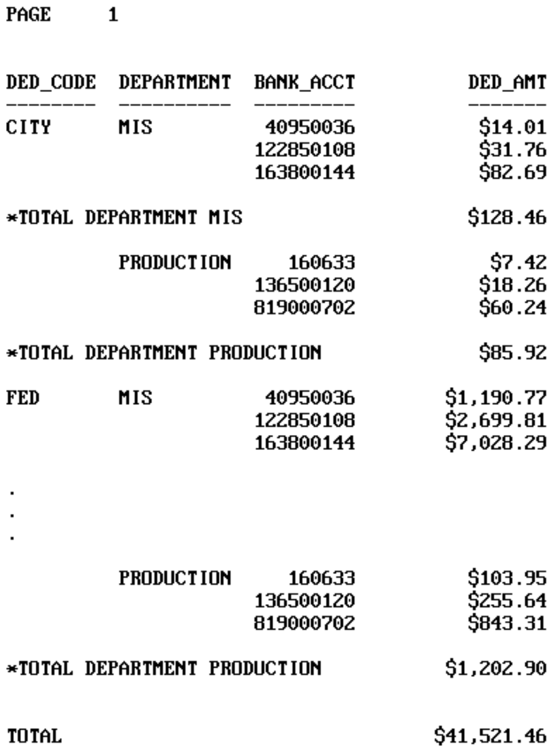
Each command produces grand totals and/or subtotals by using different numeric information. In general, subtotals produce totals every time a specified sort field value changes. Also, subtotal commands are independent of record selection phrases. To further control when subtotals are produced, use the WHEN clause (see *Conditionally Displaying Summary Lines and Text: WHEN* on page 7-21).

Example Using Section Totals and Grand Totals

You want to display subtotals and grand total amounts of deductions by department. Consider the following request:

```
TABLE FILE EMPLOYEE
SUM DED_AMT BY DED_CODE BY DEPARTMENT
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON DEPARTMENT SUBTOTAL
END
```

The first and last portions of the report are displayed:



```
PAGE     1


DED_CODE  DEPARTMENT  BANK_ACCT          DED_AMT
--------  ----------  ---------          -------
CITY      MIS          40950036           $14.01
                      122850108           $31.76
                      163800144           $82.69

*TOTAL DEPARTMENT MIS                    $128.46

          PRODUCTION     160633            $7.42
                      136500120           $18.26
                      819000702           $60.24

*TOTAL DEPARTMENT PRODUCTION              $85.92

FED       MIS          40950036        $1,190.77
                      122850108        $2,699.81
                      163800144        $7,028.29

.
.
.

          PRODUCTION     160633          $103.95
                      136500120          $255.64
                      819000702          $843.31

*TOTAL DEPARTMENT PRODUCTION           $1,202.90


TOTAL                                 $41,521.46
```

SUB-TOTAL and SUBTOTAL

Use SUBTOTAL or SUB-TOTAL to add up individual values, such as columns of numbers, each time the named sort field changes value. SUBTOTAL shows column totals each time the named sort field changes value. SUB-TOTAL shows column totals each time the named sort field and any higher-level sort fields change values.

The difference between SUB-TOTAL and SUBTOTAL is outlined below:

- The command SUB-TOTAL produces a subtotal for all numeric values when the BY field changes value and for any higher-level sort fields when their values change.
- SUBTOTAL subtotals only as the specified sort field changes value. It does not give subtotals for higher-level fields.

Both SUB-TOTAL and SUBTOTAL produce grand totals.

The subtotal is calculated every time the BY field value changes or, if a WHEN clause is used for the BY field, every time conditions in the WHEN clause are met.

The keywords to initialize the syntax are BY or ON. Use the WHEN clause to include a condition.

Syntax How to Create Subtotals

The syntax is

```
{BY|ON} fieldname   {SUB-TOTAL|SUBTOTAL}  [MULTILINES] [AS 'text']
                                          [field1 [AND] field2...]
                                          [WHEN expression;]
```

where:

fieldname
Must be the name of a field in a BY phrase. The number of fields to subtotal multiplied by the number of levels of subtotals cannot exceed 256.

MULTILINES
Is used to suppress a subtotal when there is only one value at a sort break. Since the subtotal value is equal to this one value, it is sometimes convenient to suppress it using MULTILINES. Once specified, MULTILINES suppresses the printing of a subtotal line for every sort break that has only one detail line. FOCUS also allows you to suppress grand totals using the NOTOTAL phrase, as described in *Suppressing Grand Totals: NOTOTAL* on page 7-19.

AS '*text*'
Enables you to specify a different title (see Chapter 10, *Customizing Tabular Reports*).

field1
Denotes a list of specific fields you can supply to subtotal. This list overrides the default, which includes all numeric display fields.

`WHEN` *`expression`*`;`

Specifies the conditional display of subtotals as determined by a Boolean expression (see *Conditionally Displaying Summary Lines and Text: WHEN* on page 7-21).

Reference Usage Notes for Subtotals

- Only one of the following should be used in a request: SUB-TOTAL, SUBTOTAL, or SUMMARIZE.
- When using a SUM or COUNT command with only one BY phrase in the request, SUBTOTAL and SUB-TOTAL are redundant because they will give the same result as the value of the SUM or COUNT command. However, when using a PRINT command with one BY phrase, SUBTOTAL is useful because there can be many values within a sort break.
- All subtotals are displayed up to and including the point where the sort break occurs, so that only the innermost point of subtotaling should be requested. For instance, if the BY fields are

  ```
  BY AREA
  BY PROD_CODE
  BY DATE SUB-TOTAL
  ```

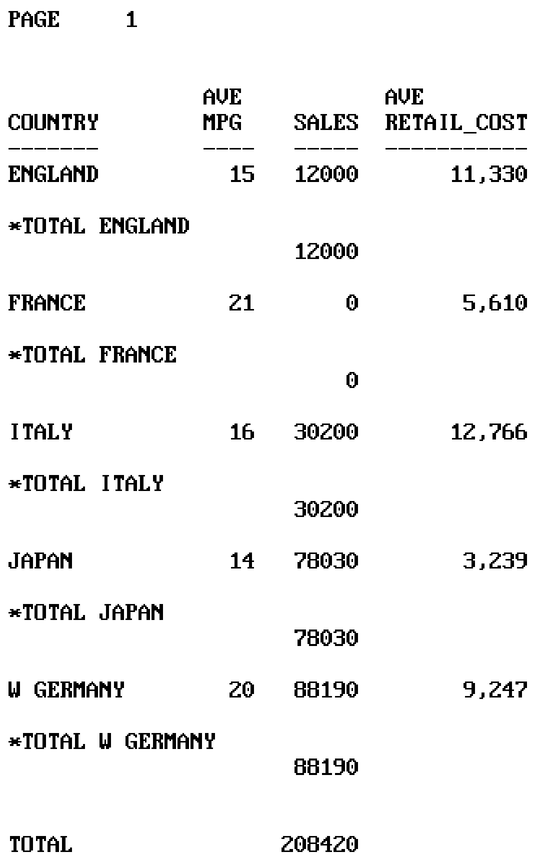
  then, when AREA changes, subtotals are displayed for AREA, PROD_CODE, and DATE on three lines (one under the other).
- To suppress grand totals produced by SUBTOTAL and SUB-TOTAL, use the NOTOTAL option (see *Suppressing Grand Totals: NOTOTAL* on page 7-19).

Example Using Subtotals

For example:

```
TABLE FILE CAR
SUM AVE.MPG AND SALES AND AVE.RETAIL_COST
BY COUNTRY SUB-TOTAL SALES
END
```

This request produces the following report:



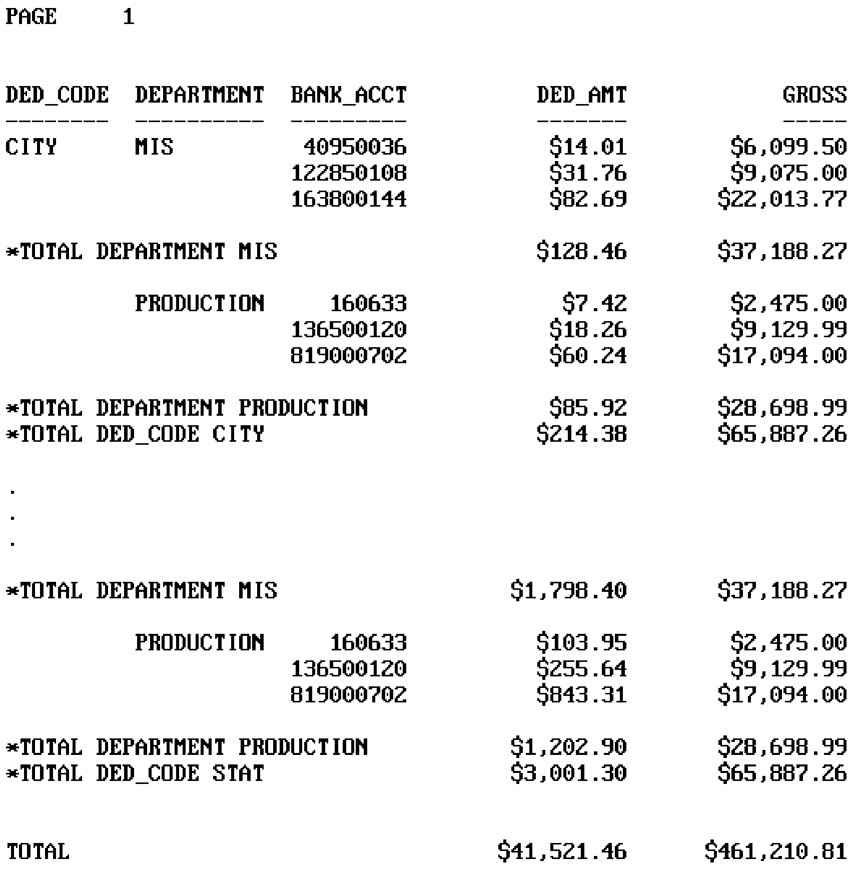
```
PAGE     1


               AVE           AVE
COUNTRY        MPG    SALES  RETAIL_COST
-------        ----   -----  -----------
ENGLAND          15   12000       11,330

*TOTAL ENGLAND
                      12000

FRANCE           21       0        5,610

*TOTAL FRANCE
                          0

ITALY            16   30200       12,766

*TOTAL ITALY
                      30200

JAPAN            14   78030        3,239

*TOTAL JAPAN
                      78030

W GERMANY        20   88190        9,247

*TOTAL W GERMANY
                      88190


TOTAL                208420
```

Example Comparing SUB-TOTAL and SUBTOTAL

Consider the following example of SUB-TOTAL,

```
TABLE FILE EMPLOYEE
SUM DED_AMT GROSS BY DED_CODE BY DEPARTMENT
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON DEPARTMENT SUB-TOTAL
END
```

which produces the following report (of which the first and last portions are shown). Note that all the BY phrase values are subtotaled. If you had used SUBTOTAL instead of SUB-TOTAL, totals for DED_AMT and GROSS would be displayed only when the DEPARTMENT field changed.



```
PAGE     1


DED_CODE  DEPARTMENT  BANK_ACCT          DED_AMT          GROSS
--------  ----------  ---------          -------          -----
CITY      MIS          40950036           $14.01      $6,099.50
                      122850108           $31.76      $9,075.00
                      163800144           $82.69     $22,013.77

*TOTAL DEPARTMENT MIS                    $128.46     $37,188.27

          PRODUCTION     160633            $7.42      $2,475.00
                      136500120           $18.26      $9,129.99
                      819000702           $60.24     $17,094.00

*TOTAL DEPARTMENT PRODUCTION              $85.92     $28,698.99
*TOTAL DED_CODE CITY                     $214.38     $65,887.26

.
.
.

*TOTAL DEPARTMENT MIS                  $1,798.40     $37,188.27

          PRODUCTION     160633          $103.95      $2,475.00
                      136500120          $255.64      $9,129.99
                      819000702          $843.31     $17,094.00

*TOTAL DEPARTMENT PRODUCTION           $1,202.90     $28,698.99
*TOTAL DED_CODE STAT                   $3,001.30     $65,887.26


TOTAL                                 $41,521.46    $461,210.81
```

SUMMARIZE and RECOMPUTE

SUMMARIZE and RECOMPUTE should be used instead of SUB-TOTAL and SUBTOTAL to recalculate computed fields.

The keywords that initialize the syntax are ON or BY. Use the WHEN clause to conditionally display subtotals for computed fields.

SUMMARIZE works like SUB-TOTAL in that it recomputes values at every sort break. In contrast, RECOMPUTE recalculates only at the specified sort break, like SUBTOTAL.

Syntax How to Subtotal Computed Fields

The syntax is

```
{BY|ON} fieldname  {RECOMPUTE|SUMMARIZE}  [MULTILINES] [AS 'text']
                                          [field1...]
                                          [WHEN expression;]
```

where:

fieldname
: Must name a field in a BY phrase. The number of fields subtotaled multiplied by the number of levels of subtotals cannot exceed 256.

MULTILINES
: When there is only one value at a sort break, the subtotal value is equal to this one value. If you wish to suppress this subtotal, include the MULTILINES phrase.

AS '*text*'
: Enables you to specify a different title (see Chapter 10, *Customizing Tabular Reports*).

field1
: Denotes a list of specific fields you can supply to be subtotaled after the RECOMPUTE or SUMMARIZE. This list overrides the default, which includes all numeric display fields.

WHEN *expression*;
: Specifies the conditional display of subtotals based on a Boolean expression (see *Conditionally Displaying Summary Lines and Text: WHEN* on page 7-21).

Reference Usage Notes for Summarizing Computed Fields

- You may also generate subtotals for the recalculated values of computed fields used in COMPUTE commands with the ON TABLE phrase. Use the following syntax:

  ```
  ON TABLE SUMMARIZE
  ```

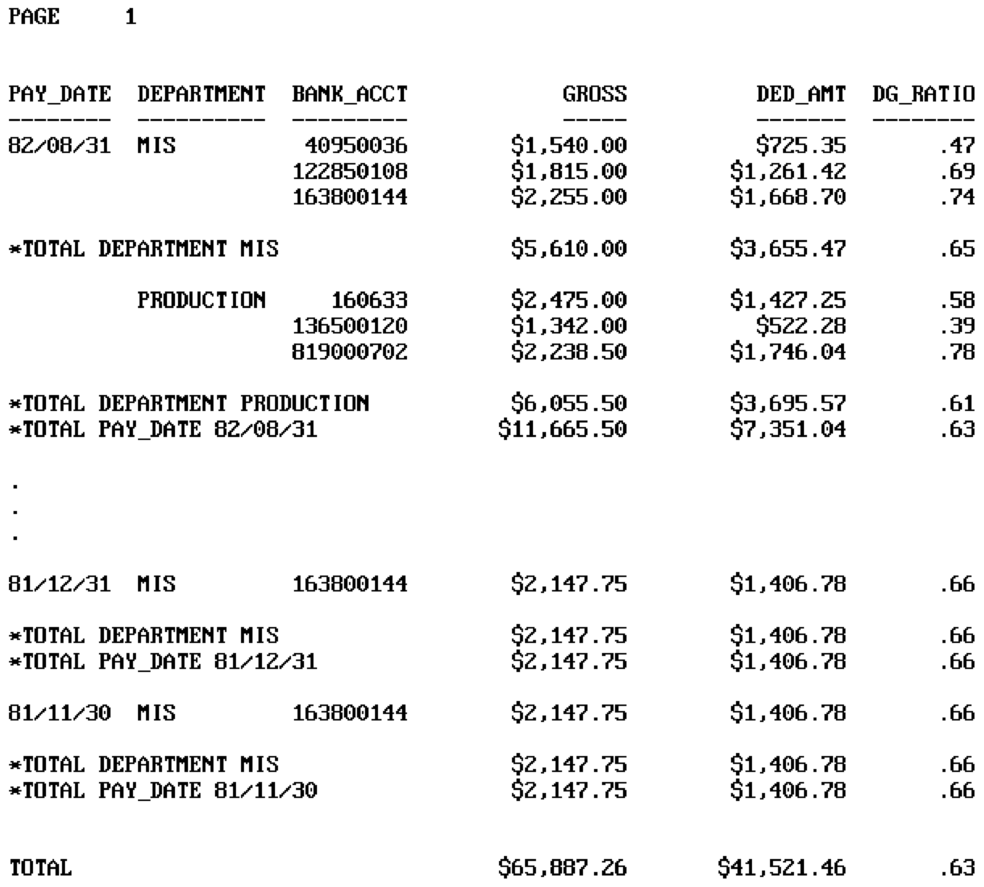
- SUMMARIZE recomputes grand totals for the entire report. To suppress grand totals, include the NOTOTAL option discussed in *Suppressing Grand Totals: NOTOTAL* on page 7-19.

Example Using SUMMARIZE

In the following example, notice how SUMMARIZE recalculates DG_RATIO:

```
TABLE FILE EMPLOYEE
SUM GROSS DED_AMT AND COMPUTE
DG_RATIO/F4.2=DED_AMT/GROSS;
BY HIGHEST PAY_DATE BY DEPARTMENT
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON DEPARTMENT SUMMARIZE
END
```

If you used the SUB-TOTAL or SUBTOTAL command, the values of DG_RATIO would simply be added. The first and last portions of the report are displayed below:



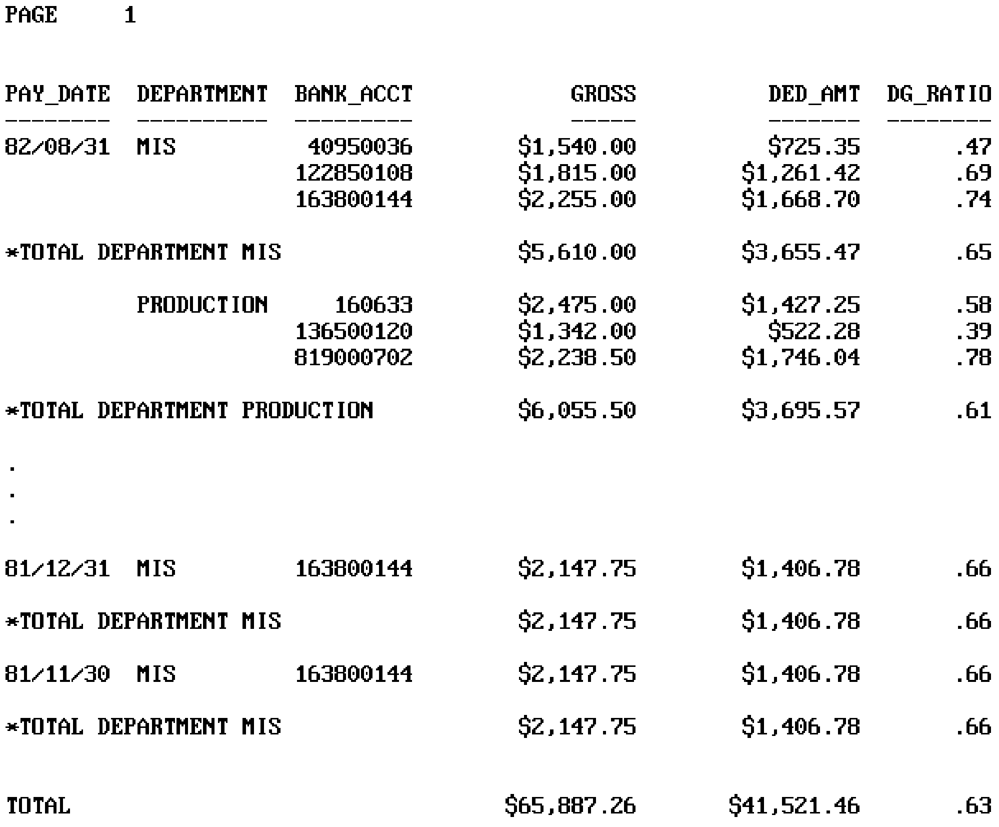
```
PAGE     1


PAY_DATE  DEPARTMENT  BANK_ACCT            GROSS          DED_AMT  DG_RATIO
--------  ----------  ---------            -----          -------  --------
82/08/31  MIS          40950036        $1,540.00          $725.35       .47
                      122850108        $1,815.00        $1,261.42       .69
                      163800144        $2,255.00        $1,668.70       .74

*TOTAL DEPARTMENT MIS                  $5,610.00        $3,655.47       .65

          PRODUCTION     160633        $2,475.00        $1,427.25       .58
                      136500120        $1,342.00          $522.28       .39
                      819000702        $2,238.50        $1,746.04       .78

*TOTAL DEPARTMENT PRODUCTION           $6,055.50        $3,695.57       .61
*TOTAL PAY_DATE 82/08/31              $11,665.50        $7,351.04       .63

.
.
.

81/12/31  MIS         163800144        $2,147.75        $1,406.78       .66

*TOTAL DEPARTMENT MIS                  $2,147.75        $1,406.78       .66
*TOTAL PAY_DATE 81/12/31               $2,147.75        $1,406.78       .66

81/11/30  MIS         163800144        $2,147.75        $1,406.78       .66

*TOTAL DEPARTMENT MIS                  $2,147.75        $1,406.78       .66
*TOTAL PAY_DATE 81/11/30               $2,147.75        $1,406.78       .66


TOTAL                                 $65,887.26       $41,521.46       .63
```

Example Using RECOMPUTE

Consider the following example of RECOMPUTE:

```
TABLE FILE EMPLOYEE
SUM GROSS DED_AMT AND COMPUTE
DG_RATIO/F4.2=DED_AMT/GROSS;
BY HIGHEST PAY_DATE BY DEPARTMENT
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON DEPARTMENT RECOMPUTE
END
```

The first and last portions of the report are displayed below:



```
PAGE     1


PAY_DATE  DEPARTMENT  BANK_ACCT             GROSS          DED_AMT  DG_RATIO
--------  ----------  ---------             -----          -------  --------
82/08/31  MIS          40950036         $1,540.00          $725.35       .47
                      122850108         $1,815.00        $1,261.42       .69
                      163800144         $2,255.00        $1,668.70       .74

*TOTAL DEPARTMENT MIS                   $5,610.00        $3,655.47       .65

          PRODUCTION     160633         $2,475.00        $1,427.25       .58
                      136500120         $1,342.00          $522.28       .39
                      819000702         $2,238.50        $1,746.04       .78

*TOTAL DEPARTMENT PRODUCTION            $6,055.50        $3,695.57       .61

.
.
.

81/12/31  MIS         163800144         $2,147.75        $1,406.78       .66

*TOTAL DEPARTMENT MIS                   $2,147.75        $1,406.78       .66

81/11/30  MIS         163800144         $2,147.75        $1,406.78       .66

*TOTAL DEPARTMENT MIS                   $2,147.75        $1,406.78       .66


TOTAL                                  $65,887.26       $41,521.46       .63
```

RECAP and COMPUTE

RECAP and COMPUTE enable you to use subtotal values in a calculation. The subtotal values are not displayed; only the result of the calculation is printed on the report.

Syntax How to Use Subtotals in Calculations

RECAP has similar syntax to other total and subtotal commands and the COMPUTE command. The syntax is

```
{ON|BY}  fieldname1  {RECAP|COMPUTE}  fieldname2[/format] = expression;
                                      [WHEN expression;]
```

where:

fieldname1
Is the field in the BY phrase. Each time the BY field changes value, there is a new recap.

RECAP
Initializes the syntax to calculate a new field for subtotal purposes. You can also use the word COMPUTE.

fieldname2
Is the new field equal to the expression.

/format
The default is D12.2.

expression
Can be any valid expression, described in Chapter 8, *Using Expressions*. RECAP and COMPUTE commands must end with semicolons.

WHEN *expression*
Is for use with RECAP only. Specifies the conditional display of RECAP lines as determined by a Boolean expression (see *Conditionally Displaying Summary Lines and Text: WHEN* on page 7-21).

Reference Usage Notes for RECAP and COMPUTE

- RECAPs automatically use either the current value of the named sort field, the current subtotal values of any computational fields that appear as display fields, or the last value for alphanumeric fields.
- The field names in the expression must be fields that appear on the report. That is, they must be display fields, or sort control fields.
- Each RECAP value is displayed on a separate line. However, if the request contains a RECAP subcommand and SUBFOOT text, the RECAP value can be displayed only in the SUBFOOT text and must be specified in the text using a spot marker (see Chapter 10, *Customizing Tabular Reports*).
- The calculations in a RECAP or COMPUTE can appear anywhere under the control break along with any text (see Chapter 10, *Customizing Tabular Reports*).
- The word RECAP may not be specified more than seven times; however, more than seven RECAP calculations are permitted. Use the following syntax:

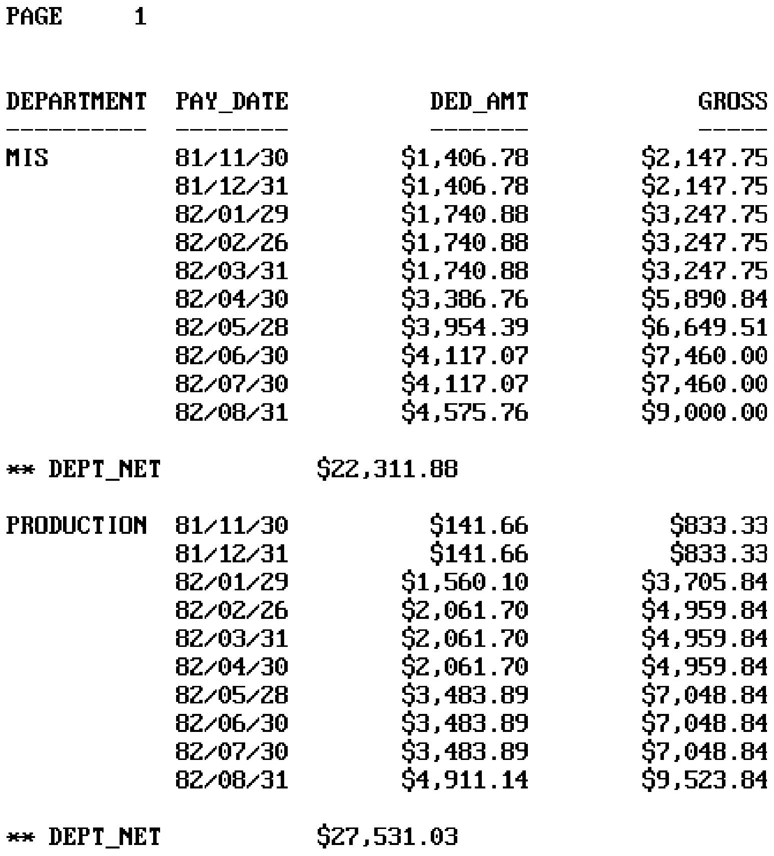
  ```
  ON fieldname RECAP field1/format= ... ;
  field2/format= ... ;
  .
  .
  .
  ```

Example Using RECAP

The following request determines net earnings for each department by subtracting the deduction amount from the gross value:

```
TABLE FILE EMPLOYEE
SUM DED_AMT AND GROSS
BY DEPARTMENT BY PAY_DATE
ON DEPARTMENT RECAP DEPT_NET/D8.2M = GROSS-DED_AMT;
WHEN PAY_DATE GT 820101
END
```

This request produces the following report. Notice the new field DEPT_NET, which holds the result of the calculation GROSS minus DED_AMT (GROSS-DED_AMT), and has a format of D8.2M.



```
PAGE     1


DEPARTMENT  PAY_DATE          DED_AMT             GROSS
----------  --------          -------             -----
MIS         81/11/30        $1,406.78         $2,147.75
            81/12/31        $1,406.78         $2,147.75
            82/01/29        $1,740.88         $3,247.75
            82/02/26        $1,740.88         $3,247.75
            82/03/31        $1,740.88         $3,247.75
            82/04/30        $3,386.76         $5,890.84
            82/05/28        $3,954.39         $6,649.51
            82/06/30        $4,117.07         $7,460.00
            82/07/30        $4,117.07         $7,460.00
            82/08/31        $4,575.76         $9,000.00

** DEPT_NET           $22,311.88

PRODUCTION  81/11/30          $141.66           $833.33
            81/12/31          $141.66           $833.33
            82/01/29        $1,560.10         $3,705.84
            82/02/26        $2,061.70         $4,959.84
            82/03/31        $2,061.70         $4,959.84
            82/04/30        $2,061.70         $4,959.84
            82/05/28        $3,483.89         $7,048.84
            82/06/30        $3,483.89         $7,048.84
            82/07/30        $3,483.89         $7,048.84
            82/08/31        $4,911.14         $9,523.84

** DEPT_NET           $27,531.03
```

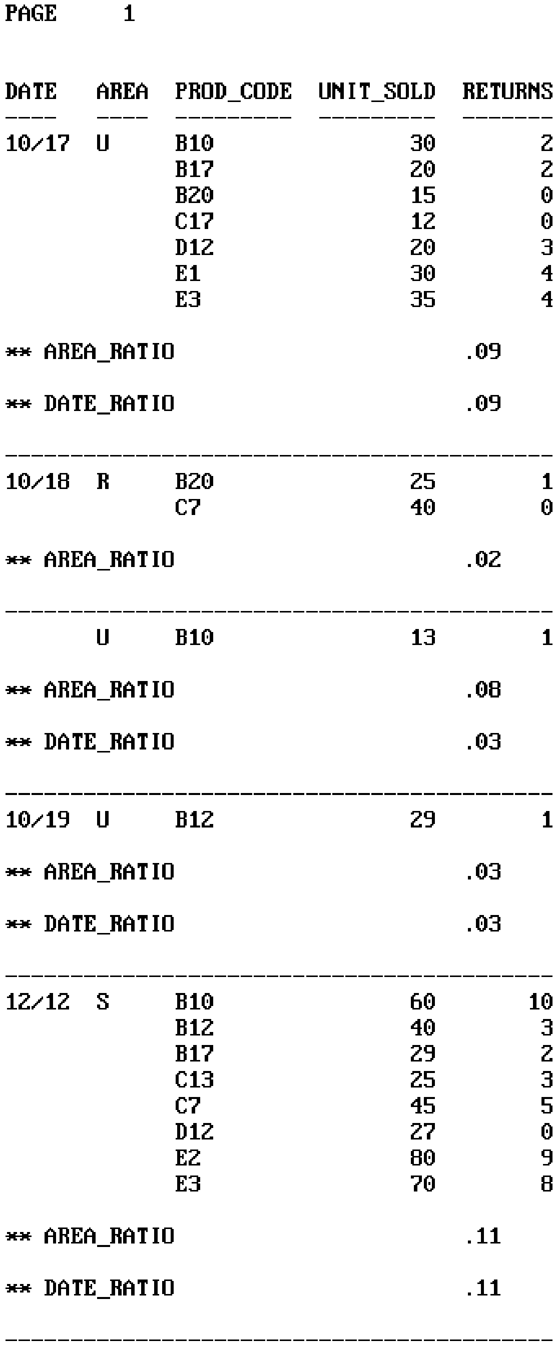
Example Using Multiple RECAPs

You can have multiple RECAP or COMPUTE subcommands in a report request. This option allows different calculations to be performed at different control breaks.

For example:

```
TABLE FILE SALES
SUM UNIT_SOLD AND RETURNS
BY DATE BY AREA BY PROD_CODE
ON DATE RECAP
DATE_RATIO=RETURNS/UNIT_SOLD;
ON AREA UNDER-LINE RECAP
AREA_RATIO=RETURNS/UNIT_SOLD;
END
```

This request produces the following report (of which the first page is shown):



```
PAGE     1


DATE   AREA  PROD_CODE  UNIT_SOLD  RETURNS
----   ----  ---------  ---------  -------
10/17  U     B10               30        2
             B17               20        2
             B20               15        0
             C17               12        0
             D12               20        3
             E1                30        4
             E3                35        4

** AREA_RATIO                      .09

** DATE_RATIO                      .09

------------------------------------------
10/18  R     B20               25        1
             C7                40        0

** AREA_RATIO                      .02

------------------------------------------
       U     B10               13        1

** AREA_RATIO                      .08

** DATE_RATIO                      .03

------------------------------------------
10/19  U     B12               29        1

** AREA_RATIO                      .03

** DATE_RATIO                      .03

------------------------------------------
12/12  S     B10               60       10
             B12               40        3
             B17               29        2
             C13               25        3
             C7                45        5
             D12               27        0
             E2                80        9
             E3                70        8

** AREA_RATIO                      .11

** DATE_RATIO                      .11

------------------------------------------
```

Suppressing Grand Totals: NOTOTAL

To further enhance your control of totaling on reports, FOCUS also enables you to suppress grand totals by using the NOTOTAL phrase.

Syntax How to Suppress Grand Totals

To suppress the grand total at the end of a report, use NOTOTAL. The syntax is:

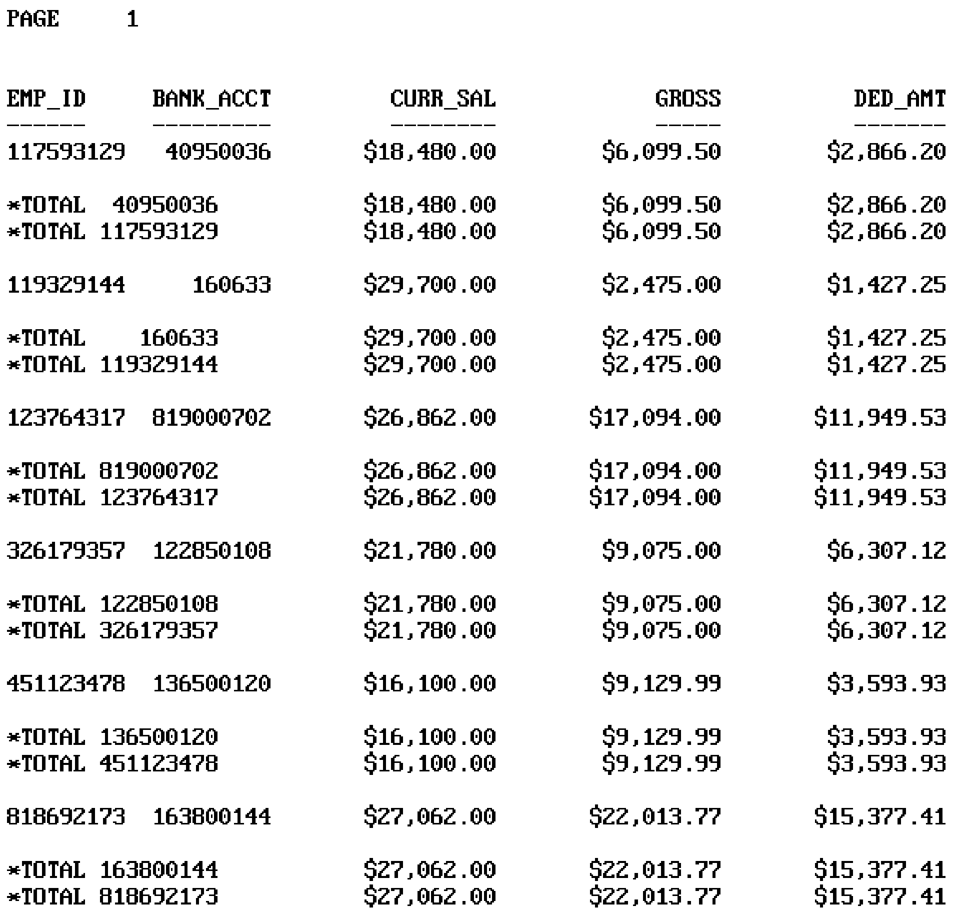
```
ON TABLE NOTOTAL
```

Example Suppressing Grand Totals

For example:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AND GROSS AND DED_AMT
BY EMP_ID
BY BANK_ACCT
WHERE BANK_ACCT NE 0
ON BANK_ACCT SUB-TOTAL
ON TABLE NOTOTAL
END
```

This request produces the following report, where the grand totals for CURR_SAL, GROSS, and DED_AMT are not displayed:



```
PAGE     1


EMP_ID     BANK_ACCT         CURR_SAL           GROSS          DED_AMT
------     ---------         --------           -----          -------
117593129   40950036       $18,480.00       $6,099.50        $2,866.20

*TOTAL  40950036           $18,480.00       $6,099.50        $2,866.20
*TOTAL 117593129           $18,480.00       $6,099.50        $2,866.20

119329144     160633       $29,700.00       $2,475.00        $1,427.25

*TOTAL    160633           $29,700.00       $2,475.00        $1,427.25
*TOTAL 119329144           $29,700.00       $2,475.00        $1,427.25

123764317  819000702       $26,862.00      $17,094.00       $11,949.53

*TOTAL 819000702           $26,862.00      $17,094.00       $11,949.53
*TOTAL 123764317           $26,862.00      $17,094.00       $11,949.53

326179357  122850108       $21,780.00       $9,075.00        $6,307.12

*TOTAL 122850108           $21,780.00       $9,075.00        $6,307.12
*TOTAL 326179357           $21,780.00       $9,075.00        $6,307.12

451123478  136500120       $16,100.00       $9,129.99        $3,593.93

*TOTAL 136500120           $16,100.00       $9,129.99        $3,593.93
*TOTAL 451123478           $16,100.00       $9,129.99        $3,593.93

818692173  163800144       $27,062.00      $22,013.77       $15,377.41

*TOTAL 163800144           $27,062.00      $22,013.77       $15,377.41
*TOTAL 818692173           $27,062.00      $22,013.77       $15,377.41
```

Conditionally Displaying Summary Lines and Text: WHEN

In addition to using summary lines to control the look and content of your report, you can control the conditions under which summary lines appear for each sort field value.

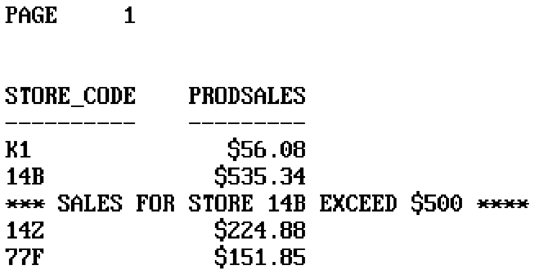
For information on using the WHEN clause to control the display of summary lines and formatting options for BY fields, see *Calculating Row and Column Totals* on page 7-1 through *RECAP and COMPUTE* on page 7-15. The WHEN clause is fully described in Chapter 10, *Customizing Tabular Reports*.

Example Conditionally Displaying Summary Lines and Text

In a sales report that covers ten stores, you might want to display a subtotal for each store but include summary lines only for stores with a minimum of $500 worth of sales. To conditionally display summary lines and text, use the WHEN clause as shown in the following request:

```
DEFINE FILE SALES
PRODSALES/D9.2M = UNIT_SOLD * RETAIL_PRICE ;
END
TABLE FILE SALES
SUM PRODSALES
BY STORE_CODE
ON STORE_CODE SUBFOOT
"*** SALES FOR STORE <STORE_CODE EXCEED $500 ****"
WHEN PRODSALES GT 500
END
```

The request produces the following report. Although the subtotal is displayed for each store, the subfoot occurs only when store sales exceed $500, as specified in the WHEN clause.



```
PAGE     1


STORE_CODE    PRODSALES
----------    ---------
K1               $56.08
14B             $535.34
*** SALES FOR STORE 14B EXCEED $500 ****
14Z             $224.88
77F             $151.85
```

8 Using Expressions

Topics:

- Using an Expression in FOCUS Commands and Phrases
- Types of Expressions You Can Write
- Writing Numeric Expressions
- Writing Date Expressions
- Writing Alphanumeric Expressions
- Writing Logical Expressions
- Conditional Expressions

An expression enables you to combine field names, constants, and operators in an operation that returns a single value. You can use an expression in a variety of FOCUS commands to assign that value to a temporary field or an amper variable, or use it in screening. In addition, you can use expressions, functions, and subroutines as building blocks for more complex expressions.

When you write an expression, you can specify the operation yourself, or you can use one of the many functions and subroutines provided with FOCUS for performing specific calculations or data manipulations. FOCUS offers a rich set of functions and subroutines that operate on one or more arguments and return a single value as a result. Functions and subroutines provide a convenient way to perform certain calculations and manipulations. To use a function or a subroutine, you simply call it. Refer to Chapter 9, *Using Functions and Subroutines,* for more information.

Using an Expression in FOCUS Commands and Phrases

You can use an expression in various FOCUS commands and phrases. An expression may not exceed 40 lines and must end with a semicolon, except in WHERE and WHEN phrases, where the semicolon is optional.

The FOCUS commands that can accept expressions, and the command syntax, are:

- DEFINE command.

  ```
  DEFINE FILE filename
      field[/format] = expression;
          .
          .
          .
          field[/format] = expression;
      END
  ```

- DEFINE attribute in a Master File.

  ```
  DEFINE field[/format] = expression;$
  ```

- COMPUTE command in a report request.

  ```
  COMPUTE field[/format] = expression;
  ```

- RECAP command in Extended Matrix Reporting (EMR).

  ```
  RECAP field[(n)][/format] = expression;
  ```

- WHERE or IF phrases to screen records in a report request.

  ```
  {WHERE|IF}logicalexpression[;]
  ```

- WHEN phrase to control printing of report headers and footers.

  ```
  WHEN logicalexpression[;]
  ```

- -IF command to determine branching in a Dialogue Manager procedure.

  ```
  -IF logicalexpression [THEN] GOTO label [ELSE GOTO label];
  ```

- -SET command to assign a value to a Dialogue Manager amper variable.

  ```
  -SET &variable = expression;
  ```

Expressions and Field Formats

When you use an expression to assign a value to a field, be sure to assign a field format that is consistent with the value returned by the expression. For example, if you use an alphanumeric expression to concatenate the first name and last name into a value for the field FULL_NAME, make sure you assign that field an alphanumeric format as in the following example:

```
DEFINE FILE EMPLOYEE
FULL_NAME/A25 = FIRST_NAME | LAST_NAME ;
END
```

Types of Expressions You Can Write

An expression can be one of the following types:

- **Numeric.** A numeric expression returns a numeric value. Use numeric expressions to perform calculations that use numeric constants or fields. For example, you can write an expression to compute the bonus for each employee by multiplying the current salary by the desired percentage as follows:

  ```
  COMPUTE BONUS = CURR_SAL * 0.05 ;
  ```

  For information on numeric expressions, see *Writing Numeric Expressions* on page 8-4.

- **Date.** A date expression returns a date or an integer that represents the number of days, months, quarters, or years between two dates. Use date expressions to perform numeric calculations that involve dates. For example, you can write an expression to determine when a customer can expect to receive an order by adding the number of days in transit to the date on which you shipped the order as follows:

  ```
  DEFINE DELIVERY/MDY = SHIPDATE + 5 ;
  ```

  For information on date expressions, see *Writing Date Expressions* on page 8-5.

- **Alphanumeric.** An alphanumeric expression returns an alphanumeric value. Use alphanumeric expressions to manipulate alphanumeric constants or fields. For example, you can write an expression to extract the first initial from an alphanumeric field as follows:

  ```
  DEFINE FIRST_INIT/A1 = EDIT (FIRST_NAME, '9$$$$$$$$$') ;
  ```

  For information on alphanumeric expressions, see *Writing Alphanumeric Expressions* on page 8-11.

- **Logical.** A logical expression returns TRUE or FALSE. Use logical expressions to determine whether a relationship between two values is true. For information on logical expressions, see *Writing Logical Expressions* on page 8-13.

- **Conditional.** A conditional expression (IF … THEN … ELSE) returns a numeric or alphanumeric value. Use conditional expressions to assign values based on the result of logical expressions. For information on conditional expressions, see *Conditional Expressions* on page 8-14.

Writing Numeric Expressions

A numeric expression performs a numeric calculation that uses numeric constants, fields, operators, functions, or subroutines to return a numeric value. When you use a numeric expression to assign a value to a field, that field must have a numeric format. The default format is D12.2.

A numeric expression can consist of:

- A numeric constant. For example,

  ```
  COMPUTE COUNT/I2 = 1 ;
  ```

- Two numeric constants or fields joined by a numeric operator. For example,

  ```
  COMPUTE BONUS = CURR_SAL * 0.05 ;
  ```

  For a list of numeric operators, see *Numeric Operators* on page 8-4.

- A numeric function or subroutine. For example,

  ```
  COMPUTE LONGEST_SIDE = MAX (WIDTH, HEIGHT, DEPTH) ;
  ```

- Two or more numeric expressions joined by a numeric operator. For example,

  ```
  COMPUTE PROFIT = (RETAIL_PRICE - UNIT_COST) * UNIT_SOLD ;
  ```

  Note the use of parentheses to change the order of evaluation of the expression. For information on the order in which FOCUS performs numeric operations, see *Order of Evaluation* on page 8-5.

FOCUS converts all numeric values to double-precision floating-point format before use in calculations, and then converts the result to the specified field format. In some cases, the conversion may result in a rounding difference.

If a number is too large (greater than 10^{75}) or too small (less than 10^{-75}), FOCUS issues an OVERFLOW or UNDERFLOW warning and displays asterisks for the field value.

Numeric Operators

The following list shows the numeric operators you can use in an expression:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
( )	parentheses

If you attempt to divide by 0, FOCUS sets the value of the expression to 0.

Order of Evaluation

FOCUS performs numeric operations in the following order:

1. Parentheses.
2. Exponentiation.
3. Division and multiplication.
4. Addition and subtraction.

Operations at the same level are performed from left to right.

The order of evaluation can affect the result of an expression. For example, the following calculation

```
COMPUTE PROFIT = RETAIL_PRICE - UNIT_COST * UNIT_SOLD ;
```

would give an incorrect result because UNIT_SOLD is first multiplied by UNIT_COST, and then the result is subtracted from RETAIL_PRICE.

You can change the order of evaluation by enclosing expressions in parentheses. An expression in parentheses is evaluated before any other expression. Using parentheses, the correct syntax for the preceding example is:

```
COMPUTE PROFIT = (RETAIL_PRICE - UNIT_COST) * UNIT_SOLD ;
```

Use parentheses to change the order of evaluation. They also may be used to improve readability.

Writing Date Expressions

A date expression performs numeric calculations that involve dates. A date expression returns a date or an integer that represents the number of days, months, quarters, or years between two dates.

A date expression can consist of:

- A date constant. For example,

  ```
  COMPUTE STARTDATE/MDYY = 'FEB 28 1999' ;
  ```

  Note the use of single quotation marks around the date constant FEB 28 1999.

- A calculation that uses numeric operators, date functions, or date subroutines to return a date. For example,

  ```
  DEFINE DELIVERY/MDY = SHIPDATE + 5 ;
  ```

- A calculation that uses numeric operators, date functions, or date subroutines to return an integer that represents the number of days, months, quarters, or years between two dates. For example,

  ```
  COMPUTE TURNAROUND/I4 = MDY (ORDERDATE, SHIPDATE) ;
  ```

Field Formats for Date Values

FOCUS enables you to work with dates in one of two ways:

- In date format. FOCUS treats the value as an integer that represents the number of days between the date value and the base date. FOCUS has two base dates for smart date formats. Format YMD fields have a base date of 12/31/1900. Format YM, YQ fields have a base date of 1/1901. When displaying the value, FOCUS converts the number to the corresponding date in the format specified for the field either in the Master File or in the command that uses an expression to assign a value to the field. These are referred to as smart date formatted fields.
- In integer, packed, or alphanumeric format with date edit options. FOCUS treats the value as an integer, a packed-decimal, or an alphanumeric string. When displaying the value, FOCUS formats it as a date. These are referred to as old date formatted fields.

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

The following table illustrates how the field format affects storage and display:

	Date Format **(For example: MDYY)**		**Integer, Packed, or Alphanumeric Format** **(For example: A8MDYY)**	
Value	**Stored**	**Displayed**	**Stored**	**Displayed**
February 28, 1999	35853	02/28/1999	02281999	02/28/1999
March 1, 1999	35854	03/01/1999	03011999	03/01/1999

Performing Calculations on Dates

The format of a field determines how you can use it in a date expression. Calculations on dates in date format can incorporate numeric operators as well as numeric functions and subroutines. Calculations on dates in integer, packed, or alphanumeric format require the use of date functions and subroutines; numeric operators would return an error message or an incorrect result.

To illustrate this point, consider the following example, which calculates how many days it takes for your shipping department to fill an order by subtracting the date on which an item is ordered, ORDERDATE, from the date on which it is shipped, SHIPDATE:

```
COMPUTE TURNAROUND/I4 = SHIPDATE - ORDERDATE ;
```

An item ordered on October 31, 1996 and shipped on November 1, 1996 should result in a difference of 1 day. The following table shows how the field format affects the result:

	Value in Date Format	**Value in Numeric Format**
SHIPDATE = March 1, 1999	35854	03011999
ORDERDATE = February 28, 1999	35853	02281999
TURNAROUND	1	730000

To obtain the correct result using fields in integer, packed, or alphanumeric format, you can use a date function, MDY, which returns the difference between two dates in the form month-day-year. Using the MDY function, you can calculate TURNAROUND as follows:

```
COMPUTE TURNAROUND/I4 = MDY (ORDERDATE, SHIPDATE) ;
```

FOCUS provides a rich set of functions and subroutines that enable you to manipulate dates in integer, packed-decimal, or alphanumeric format. For information on date functions, see Chapter 9, *Using Functions and Subroutines.*

Cross-Century Dates in DEFINE and COMPUTE

FOCUS allows century processing against virtual fields created by DEFINE and COMPUTE statements. A DEFINE can also be coded in a Master File, and century processing attributes can also be applied in this context. At the virtual field level, two Master File parameters (DEFCENT and YRTHRESH) provide a means of interpreting the century if the first two digits of the year (YYyy) are not provided elsewhere. If the first two digits are provided, they are simply accepted and validated.

Virtual field-level use of these parameters overrides and takes precedence over any application-, file-, or field-level setting as described in the *Describing Data* manual, but just for the virtual field in which they are used. This means that applications that have situations outside the application norm can still have global control and use of century processing.

The syntax for COMPUTE and DEFINE is

```
field/format  [{DEFCENT|DFC}  cc  {YRTHRESH|YRT} yy] [MISSING...] = exp;
```

where:

field
Is the name of the virtual field being created.

format
Is a date data type for the virtual field being created.

cc
Is the default century value to be applied.

yy
Is the threshold value used for application of the century value.

exp
Is the logic, routine, or constant being evaluated for the virtual field.

Note:

- When DEFCENT is used in combination with YRTHRESH, it establishes a 100 year window. Any two digit year that falls within that span has the century digits calculated as DFC. Any two digit year outside that span has the century digits calculated as DFC+1.
- The Default Century and Year Threshold values are applied to the final result of the right-hand side of an expression. Settings that individual fields within an expression may have are ignored. Thus, complex expressions in which this is not the desired behavior should use the technology to convert the fields on-the-fly to virtual Smart Date fields and then perform computation using the Smart Date versions of the fields.

The DEFCENT and YRTHRESH parameters may be abbreviated to DFC and YRT, respectively. For a complete discussion of cross-century data processing, see the *Developing Applications* manual.

Selecting the Format of the Result Field

A date expression always returns a number. That number may represent a date or the number of days, months, quarters, or years between two dates. When you use a date expression to assign a value to a field, the format you give to the field determines how the result will be displayed.

Consider the following commands. The first command calculates how many days it takes for your shipping department to fill an order by subtracting the date on which an item is ordered, ORDERDATE, from the date on which it is shipped, SHIPDATE. The second calculates a delivery date by adding 5 days to the date on which the order is shipped, SHIPDATE.

```
COMPUTE TURNAROUND/I4 = SHIPDATE - ORDERDATE ;
DEFINE DELIVERY/MDY = SHIPDATE + 5 ;
```

In the first command, the date expression returns the *number* of days it takes to fill an order; therefore, the associated field, TURNAROUND, must have an integer format. In the second command, the date expression returns the *date* on which the item will be delivered; therefore, the associated field, DELIVERY, must have a date format.

Using a Date Constant in an Expression

When you use a date constant in a calculation with fields in date format, you must enclose it in single quotation marks ('); otherwise, FOCUS interprets it as the number of days between the constant and the base date (December 31, 1900 or January 1, 1901). The following example shows how to initialize STARTDATE with the date constant 02/28/99:

```
COMPUTE STARTDATE/MDY = '022899' ;
```

The following example calculates the number of days elapsed since January 1, 1999:

```
DEFINE YEARTODATE/I4 = CURR_DATE - 'JAN 1 1999' ;
```

Extracting a Date Component

Date components include days, months, quarters, or years. You can write an expression that extracts a component from a field in date format. The following example shows how you can extract a month from SHIPDATE, which has the format MDY:

```
DEFINE SHIPMONTH/M = SHIPDATE ;
```

If SHIPDATE has the value February 29, 1999, the above expression returns the value 2 for SHIPMONTH.

Calculations on date components automatically produce a valid value for the desired component. For example, if the current value of SHIPMONTH is 2, the following expression

```
COMPUTE ADDTHREE/M = SHIPMONTH + 3 ;
```

correctly returns the value 5. If the addition of months results in an answer greater than 12, the months are adjusted correctly (for example, 11 + 3 is 2, not 14).

You cannot write an expression that extracts days, months, or quarters from a date that did not have these components. For example, you cannot extract a month from a date in YY format, which represents only the number of years.

Combining Fields With Different Components in an Expression

When using fields in date format, you can combine fields with a different order of components within the same expression. For example, consider the following two fields: DATE_PAID has the format YYMD and DUE_DATE has the format MDY. You can combine these two fields in an expression to calculate the number of days that a payment is late as follows:

```
COMPUTE DAYS_LATE/I4 = DATE_PAID - DUE_DATE ;
```

In addition, you can assign the result of a date expression to a field with a different order of components from the fields in the expression. For example, consider the field DATE_SOLD, which contains the date on which an item is sold, in YYMD format. You can write an expression that adds 7 days to DATE_SOLD to determine the last date on which the item can be returned, and then assign the result to a field with DMY format, as in the following COMPUTE command:

```
COMPUTE RETURN_BY/DMY = DATE_SOLD + 7 ;
```

You cannot, however, write an expression that combines dates in date format with dates in integer, packed, or alphanumeric format.

Writing Alphanumeric Expressions

An alphanumeric expression manipulates alphanumeric constants, fields, concatenation operators, functions, or subroutines to return an alphanumeric value. When you use an alphanumeric expression to assign a value to a field, you must give that field an alphanumeric format.

An alphanumeric expression can consist of:

- An alphanumeric constant (character string) enclosed in single quotation marks. For example,

  ```
  COMPUTE STATE = 'NY' ;
  ```

- Two or more alphanumeric fields or constants joined by the concatenation operator. For example,

  ```
  DEFINE TITLE/A19 = 'DR. ' | LAST_NAME ;
  ```

- An alphanumeric function or subroutine. For example,

  ```
  DEFINE INITIAL/A1 = EDIT (FIRST_NAME, '9$$$$$$$$$$') ;
  ```

Using Quotation Marks Within Quote-Delimited Literal Strings

Under certain conditions, FOCUS allows users to use quote-delimited strings containing embedded quotation marks. FOCUS treats two contiguous quotes within a quote-delimited string as a single literal quote.

FOCUS supports quote delimited strings in the following:

- IF and WHERE statements containing multiple quotes.
- WHERE statements containing: field {IS, IS-NOT, IN, IN FILE or NOT IN FILE}.
- EDIT.
- WHEN field EQ embedded quote in a literal.
- DEFINE statements.
- DEFINE statements in Master Files.
- DBA expressions in Master Files (for example, VALUE = IF field EQ embedded quotes in literal).
- ACCEPT=, DESCRIPTION=, TITLE= in Master Files.
- AS.
- DECODE.

Example Using Quote-Delimited Literal Strings

For example, if a report request contains the following statement:

```
IF AIRPORT EQ 'O''HARE'
```

FOCUS will interpret the literal string ‘O’’HARE and look for a data source value of O’HARE.

Concatenating Character Strings

You can write an expression to concatenate two or more alphanumeric constants or fields into a single character string. The concatenation operator takes one of two forms, as shown in the following table:

<table>
<tr><th>Symbol</th><th>Represents</th><th>Function</th></tr>
<tr><td>|</td><td>Weak concatenation</td><td>Preserves trailing blanks.</td></tr>
<tr><td>||</td><td>Strong concatenation</td><td>Suppresses trailing blanks.</td></tr>
</table>

The following example shows how to use the EDIT function to extract the first initial from a first name, and then use both strong and weak concatenation to produce the last name, followed by a comma, followed by the first initial, followed by a period:

```
DEFINE FILE EMPLOYEE
FIRST_INIT/A1 = EDIT (FIRST_NAME, '9$$$$$$$$$');
NAME/A19 = LAST_NAME ||(', '| FIRST_INIT |'.');
END
```

Suppose that FIRST_NAME has the value Chris and LAST_NAME has the value Edwards. The above request evaluates the expressions as follows:

1. The EDIT function extracts the initial C from FIRST_NAME.

2. The expression in parentheses is evaluated. It returns the value

   ```
   , C.
   ```

3. LAST_NAME is concatenated to the string derived in step 2 to produce

   ```
   Edwards, C.
   ```

   Note that while LAST_NAME has the format A15, strong concatenation suppresses the trailing blanks. The resulting field name is still alpha 19 long.

Writing Logical Expressions

A logical expression determines whether a particular condition is true. There are two kinds of logical expressions: relational and boolean. The entities you want to compare determine the kind of expression you use.

A relational expression returns TRUE or FALSE based on a comparison of two individual values (either fields or constants). A boolean expression returns TRUE or FALSE based on the outcome of two or more relational expressions.

You can use a logical expression to assign a value to a numeric field. If the expression is true, the field receives the value 1. If the expression is false, the field receives the value 0.

Relational Expressions

A relational expression returns TRUE or FALSE based on a comparison of two individual values (either fields or constants). You can use the following relational operators in an expression:

```
value {EQ|NE}  value
value {GE|GT}  value
value {LE|LT}  value
```

```
alphanumeric value {CONTAINS|OMITS} alphanumeric value
```

You must enclose alphanumeric literals with embedded blanks in single quotation marks (').

For information on relational operators and additional operators that are available for record selection using WHERE and IF, see Chapter 5, *Selecting Records for Your Report*.

Boolean Expressions

A boolean expression returns TRUE or FALSE based on the outcome of two or more relational expressions. You can use the following boolean operators in an expression:

```
(relational expression) {AND|OR} (relational expression)
NOT (logical expression)
```

Note that you must enclose expressions in parentheses.

Conditional Expressions

A conditional expression assigns a value based on the result of a logical expression. The assigned value can be numeric or alphanumeric. A conditional expression takes the following form:

```
IF expression1 THEN expression2 [ELSE expression3]
```

Note:

- The expressions following THEN and ELSE must result in a format that is compatible with the format assigned to the field. Each of these expressions may itself be a conditional expression, although the expression following IF may not be an IF … THEN … ELSE expression (for example, IF … IF …). If the expression following THEN is itself a conditional expression, it must be enclosed in parentheses, but after ELSE, this is not necessary.
- Conditional expressions can have up to 16 IF phrases.
- The keyword ELSE, followed by an expression, is optional. If you do not specify an ELSE condition, however, and the IF condition is not met, the value is taken from the last true condition.

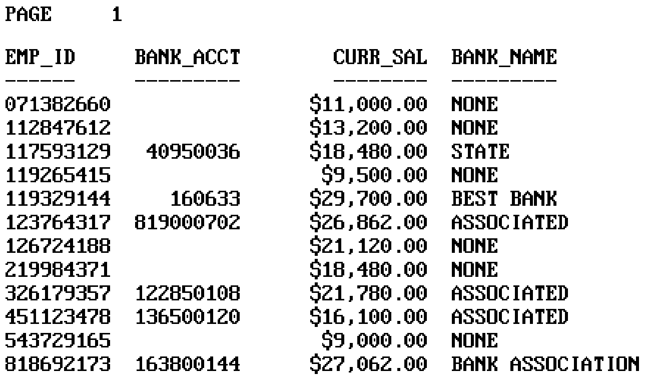
Example Supplying Values With Conditional Expressions

For example:

```
DEFINE FILE EMPLOYEE
BANK_NAME/A20 = IF BANK_NAME EQ ' ' THEN 'NONE'
ELSE BANK_NAME;
END
```

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND BANK_NAME
BY EMP_ID BY BANK_ACCT
END
```

This request produces the following report:



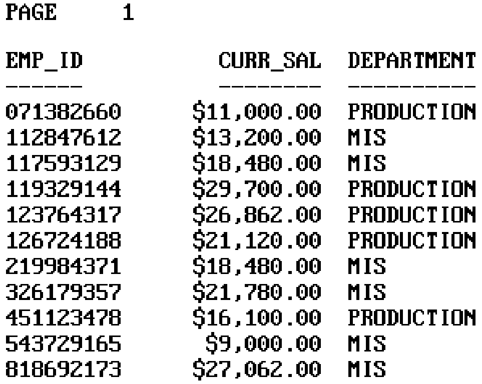
```
PAGE     1

EMP_ID     BANK_ACCT       CURR_SAL  BANK_NAME
------     ---------       --------  ---------
071382660                $11,000.00  NONE
112847612                $13,200.00  NONE
117593129   40950036     $18,480.00  STATE
119265415                 $9,500.00  NONE
119329144     160633     $29,700.00  BEST BANK
123764317  819000702     $26,862.00  ASSOCIATED
126724188                $21,120.00  NONE
219984371                $18,480.00  NONE
326179357  122850108     $21,780.00  ASSOCIATED
451123478  136500120     $16,100.00  ASSOCIATED
543729165                 $9,000.00  NONE
818692173  163800144     $27,062.00  BANK ASSOCIATION
```

Example Defining True or False Conditions

You can also define a true condition and then refer to it as TRUE or FALSE. For example:

```
DEFINE FILE EMPLOYEE
MYTEST=(CURR_SAL GE 11000) OR (DEPARTMENT EQ 'MIS');
END

TABLE FILE EMPLOYEE
PRINT CURR_SAL AND DEPARTMENT
BY EMP_ID
IF MYTEST IS TRUE
END
```

This request produces the following report:



```
PAGE     1

EMP_ID          CURR_SAL  DEPARTMENT
------          --------  ----------
071382660     $11,000.00  PRODUCTION
112847612     $13,200.00  MIS
117593129     $18,480.00  MIS
119329144     $29,700.00  PRODUCTION
123764317     $26,862.00  PRODUCTION
126724188     $21,120.00  PRODUCTION
219984371     $18,480.00  MIS
326179357     $21,780.00  MIS
451123478     $16,100.00  PRODUCTION
543729165      $9,000.00  MIS
818692173     $27,062.00  MIS
```

Note: TRUE and FALSE conditions are valid only in the IF phrase.

9 Using Functions and Subroutines

Topics:

- What Is the Difference Between a Function and a Subroutine?
- Types of Functions and Subroutines
- Subroutine Command (Call) Syntax
- Storing and Accessing External Subroutines
- Alphabetical List of Functions and Subroutines

FOCUS offers a rich set of functions and subroutines that operate on one or more arguments and return a single value as a result. Functions and subroutines provide a convenient way to perform certain calculations and manipulations.

The next few sections describe the following:

- What is the difference between a function and a subroutine?
- Types of functions and subroutines.
- How to use subroutines.
- An alphabetical description of the functions and subroutines.

You can also create your own subroutines.

What Is the Difference Between a Function and a Subroutine?

There are two differences between a function and a subroutine:

- How they are invoked.
- How they are accessed.

A function call has the following syntax

```
function (arg1, arg2, ...)
```

where:

`function`

Is the name of the function.

`arg1, arg2, ...`

Are the arguments.

A subroutine call has the following syntax

```
subroutine (arg1, arg2, ... {outputfield|'format'})
```

where:

subroutine
Is the name of the subroutine.

arg1, *arg2*, ...
Are the arguments.

{*outputfield*|'*format*'}
Is the name of the output field or its format.

In addition, on some platforms, the functions are available immediately; whereas, the subroutines are available in a special subroutine library that you must access.

Types of Functions and Subroutines

You can access any of the following kinds of functions and subroutines:

Bit
Enable you to manipulate bits.

Character
Enable you to manipulate alphanumeric fields or character strings.

Data Source
Enable you to search for or retrieve data source records or values.

Date and Time
Enable you to manipulate dates and times.

Decoding
Enable you to assign values.

Format Conversion
Enable you to convert fields from one format to another.

Numeric
Enable you to perform numeric calculations on numeric constants and fields.

System
Enable you to make calls to the operating system to obtain information about the operating environment or to use a system service.

Bit Functions and Subroutines

Bit functions and subroutines enable you to manipulate bits. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 9-31.

BITSON subroutine

Evaluates an individual bit within a character string to determine whether it is on or off.

Available on: MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

BITVAL subroutine

Evaluates a string of bits within character strings and returns its binary value.

Available on: MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

BYTVAL subroutine

Translates a character to its corresponding ASCII code.

HEXBYT subroutine

Translates an integer between 0 and 255 (base 10) into the corresponding ASCII or EBCDIC character (depending on your platform).

UFMT subroutine

Converts characters in alphanumeric field values to hexadecimal (HEX) representation.

Available on: MVS, VM/CMS, OpenVMS, and WebFOCUS.

Character Functions and Subroutines

The following functions and subroutines enable you to manipulate alphanumeric fields or character strings. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 9-31.

ARGLEN subroutine

Measures the length of a character string within a field, excluding trailing blanks.

ASIS function

In Dialogue Manager, distinguishes between a blank and a zero.

Available on: MVS, VM/CMS, and WebFOCUS.

BITSON subroutine

Evaluates an individual bit within a character string to determine whether it is on or off.

Available on: MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

BITVAL subroutine

Evaluates a string of bits within character strings and returns its binary value.

Available on: MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

BYTVAL subroutine

Translates a character to its corresponding ASCII code.

CHKFMT subroutine

Checks for incorrect character types by comparing each character in the input string to the corresponding character in a mask.

CTRAN subroutine

Converts one character in a string to another character.

CTRFLD subroutine

Centers a character string within a field, excluding trailing blanks.

EDIT function

Extracts characters or adds characters to an alphanumeric string (with mask).

GETTOK subroutine

Divides a character string at a character called the delimiter and returns a substring called the token.

LCWORD subroutine

Converts the letters in the given string to mixed case. The subroutine converts to uppercase the first letter of each new word and the first letter after a single or double quotation mark.

Available on: MVS, VM/CMS, WebFOCUS, and Windows.

LJUST subroutine

Left-justifies a character string within a field. All leading blanks become trailing blanks.

LOCASE subroutine

Converts uppercase characters to lowercase.

OVRLAY subroutine

Overlays a substring on another character string.

PARAG subroutine

Divides lines of text into smaller lines with delimiters.

POSIT subroutine

Finds the starting position of a substring within a larger string.

REVERSE subroutine

Inverts and prints the contents of a field.

Available on: UNIX, OpenVMS, and Windows.

RJUST subroutine

Right-justifies a character string within a field. All trailing blanks become leading blanks.

SIMILAR subroutine

Matches two character strings for a degree of likeness on a scale of 1 to 100.

Available on: WebFOCUS and Windows.

SOUNDEX subroutine

Searches for a character string phonetically rather than by the way it is spelled.

SPELLNUM subroutine

Takes an alphanumeric string or a numeric value with two decimal places and spells it out with dollars and cents.

Available on: Windows.

SUBSTR subroutine

Extracts substrings, based on where they start and end in the parent string.

UPCASE subroutine

Converts lowercase characters to uppercase.

Data Source Functions and Subroutines

Data source functions and subroutines enable you to search for or retrieve data source records or values. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 9-31.

FIND function

Verifies if a value exists in an indexed field in another file.

Available on: MVS, VM/CMS, and UNIX.

LAST function

Retrieves the preceding value selected for a field.

LOOKUP function

Retrieves a value from a cross-referenced file.

Available on: MVS, VM/CMS, and UNIX.

Date Functions and Subroutines

The following functions and subroutines enable you to manipulate dates. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 9-31.

AYM subroutine

Adds or subtracts months from dates that are in year-month format.

AYMD subroutine

Adds or subtracts days from dates that are in year-month-day format.

CHGDAT subroutine

Rearranges the year, month, and day portions of dates, and converts dates between long and short date formats.

DA subroutines

Convert dates to the corresponding number of days elapsed since December 31, 1899.

DATEADD subroutine

Adds or subtracts date units to or from a date.

Available on: MVS and VM/CMS.

DATEDIF function

Calculates the difference between two dates.

Available on: MVS and VM/CMS.

DATECVT function

Converts dates from one date format to another.

Available on: MVS and VM/CMS.

DATEMOV function

Moves a date to a significant point on the calendar.

Available on: MVS and VM/CMS.

DMY, MDY, and YMD functions

Calculate the difference between two dates.

DOWK[L] subroutines

Determine the day of the week for dates.

DT subroutines

Convert smart dates (the number of days elapsed since December 31, 1899) to corresponding dates.

GREGDT subroutine

Converts dates in Julian format to year-month-day format.

HHMMSS subroutine

Retrieves the current time from the system.

JULDAT subroutine

Converts dates from year-month-day format to Julian (year-day format).

TODAY subroutine

Retrieves the current date from the system.

YM subroutine

Calculates the number of months that elapse between two dates. The dates must be in year-month format.

Valid Date Input

Date subroutines accept the following types of dates:

- Years that are four digits and display the century, such as 2000 and 1900, if their formats are specified as I8YYMD, P8YYMD, D8YYMD, F8YYMD, or A8YYMD.

  The following example uses the DECODE function to assign dates with four-digit years. It then converts these dates to Julian and Gregorian formats.

  ```
  DEFINE FILE CAR
  DATE/I8YYMD=DECODE COUNTRY (ENGLAND 19960101 FRANCE 19991231 ELSE 20010101);
  JDATE/I7=JULDAT(DATE,'I7');
  GDATE/I8=GREGDT(JDATE,'I8');
  END
  TABLE FILE CAR
  PRINT DATE JDATE GDATE
  END
  ```

  The request produces the following report:

  ```
  PAGE     1


        DATE    JDATE     GDATE
       ----    -----     -----
  2001/01/01  2001001  20010101
  1996/01/01  1996001  19960101
  2001/01/01  2001001  20010101
  2001/01/01  2001001  20010101
  2001/01/01  2001001  20010101
  1999/12/31  1999365  19991231
  ```

- Two-digit years with a field length of 6 (such as I6YMD). In this case, you can use the SET DEFCENT and SET YRTHRESH commands to assign the century values.

  The following example shows how to return an eight-digit date from the AYMD subroutine when the input argument has a six-digit date format. Since DEFCENT is 19 and YRTHRESH is 50, year values from 50 through 99 are interpreted as 1950 through 1999, and year values from 00 through 49 are interpreted as 2000 through 2049:

  ```
  SET DEFCENT=19, YRTHRESH=50
  TABLE FILE DATE
  PRINT D2_I6YMD AND COMPUTE
  NEWDATE/I8YYMD=AYMD(D2_I6YMD,1,'I8');
  END
  ```

  The DEFCENT and YRTHRESH values create a 100-year window as follows:

  ```
   0 ______________________ < YRTHRESH=50 ≥ ____________________ 99
               ⇑                                     ⇑
     Century=DEFCENT+1 (20)               Century=DEFCENT (19)
  ```

The request produces the following report:

```
PAGE     1

D2_I6YMD     NEWDATE
--------     ----------
97/09/16     1997/09/17
00/02/29     2000/03/01
01/02/28     2001/03/01
00/02/28     2000/02/29
```

Note: If you do not require dates for the year 2000 and beyond, you can deactivate this feature by issuing the following command:

```
SET DATEFNS=OFF
```

Syntax

How to Activate Subroutines That Support Dates for the Year 2000

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

where:

`ON`

Activates the subroutines that support dates for the year 2000 and beyond. This value is the default.

`OFF`

Deactivates the subroutines that support dates for the year 2000 and beyond; this is useful if you require the older version that does not have Year 2000 capabilities.

Leading Zeros

Use this feature in Dialogue Manager for date subroutines that return a numeric integer format. This feature 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

How to Set the Display of 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. This is the default.

Example Displaying 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);
```

Decoding Functions and Subroutines

Decoding functions and subroutines enable you to assign values: Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 9-31.

DECODE function
 Assigns values based on the value of an input field.

Format Conversion Functions and Subroutines

The following functions and subroutines convert fields from one format to another. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 9-31.

ASIS function

In Dialogue Manager, distinguishes between a blank and a zero.

Available on: MVS, VM/CMS, and WebFOCUS.

ATODBL subroutine

Converts a number in alphanumeric format to double-precision format.

CHKPCK subroutine

Verifies that the value of a packed field is indeed in packed format.

Available on: MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

EDIT function

Converts an alphanumeric field to numeric or a numeric field to alphanumeric.

FTOA subroutine

Converts a number in a numeric format to alphanumeric format.

ITONUM subroutine

Converts large binary integers in non-FOCUS files to double-precision format.

Available on: MVS, VM/CMS, and WebFOCUS.

ITOPACK subroutine

Converts large binary integers in non-FOCUS files to packed-decimal format.

Available on: MVS, VM/CMS, and WebFOCUS.

ITOZ subroutine

Converts numbers from numeric format to zoned format for extract files.

Available on: MVS, VM/CMS, OpenVMS, and WebFOCUS.

PCKOUT subroutine

Writes packed numbers of varying lengths (between one and 16 bytes) to extract files.

Available on: MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

UFMT subroutine

Converts characters in alphanumeric field values to hexadecimal (HEX) representation.

Available on: MVS, VM/CMS, OpenVMS, and WebFOCUS.

Numeric Functions and Subroutines

The following functions and subroutines enable you to perform numeric calculations on numeric constants or fields. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 9-31.

ABS function

Returns the absolute value of its argument.

ASIS function

In Dialogue Manager, distinguishes between a blank and a zero.

Available on: MVS, VM/CMS, and WebFOCUS.

BAR subroutine

Produces horizontal bar charts in reports.

Available on: MVS and VM/CMS.

EXP subroutine

Raises the number “e” to a power you specify.

Available on: MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

EXPN function

Evaluates an argument expressed in scientific notation.

Available on: MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

IMOD, DMOD, and FMOD

Calculate the remainder from a division and returns it in integer format.

INT function

Returns the integer part of its argument.

LOG function

Returns the logarithm of its argument.

MAX and MIN functions

Returns the maximum or minimum value from its list of arguments.

PRDNOR, PRDUNI, RDNORM, and RDUNIF subroutines

Generate random numbers.

ROUND subroutine

Rounds a floating-point number.

Available on: UNIX and Windows.

SQRT function

Returns the square root of its argument.

System Functions and Subroutines

The following functions and subroutines enable you to make calls to the operating system to obtain information about the operating environment or to use a system service. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 9-31.

CALLDOS subroutine

Calls a DOS program, a DOS batch program, or a Windows application.

Available on: Windows.

FEXERR subroutine

Retrieves error messages.

Available on: MVS and VM/CMS.

FINDMEM subroutine

Determines if a specific member of a partitioned data set exists.

Available on: MVS.

FPUTENV subroutine

Assigns a character string to an environment variable.

Available on: UNIX.

GETPDS subroutine

Determines if a specific member of a partitioned data set exists, and if so, returns the data set name.

Available on: MVS.

GETUSER subroutine

Retrieves the user ID from the system.

Available on: MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

HHMMSS subroutine

Retrieves the current time from the system.

ISALIVE subroutine

Tests whether another process is running and returns the Microsoft Windows Module Handle number.

Available on: Windows.

MVSDYNAM subroutine

Passes a DYNAM command to the command processor.

Available on: MVS.

SPACE subroutine

Determines whether a particular disk drive is ready and the number of bytes of disk space available.

Available on: Windows.

SPAWN subroutine

Spawns a child process in order to execute system commands without terminating the current procedure. Once the child process terminates, control returns to the parent process.

Available on: UNIX.

TODAY subroutine

Retrieves the current date from the system.

Subroutine Command (Call) Syntax

This section describes the general syntax for subroutine calls, types of arguments, and rules for using arguments.

Subroutines return a single value or character string which can be stored in a field, assigned to a Dialogue Manager variable, used in calculations and other processing, or used in selection or validation tests. Subroutine calls in FOCUS commands have the general syntax

```
subroutine (input1, input2, input3, ... {outfield|'format'}
```

where:

subroutine

Is the name of the subroutine, up to eight characters long.

input1...

Are the input subroutine arguments (data values and fields that the subroutine needs to do its processing).

{*outfield*|'*format*'}

Is the output argument. It is the name of the field that contains the output or the format of the output value, enclosed in single quotation marks, depending on the application. For Maintain, specify the field name. Maintain does not support the output format as an argument. Dialogue Manager requires the output format.

The basic syntax to store the output in a field looks like:

```
field = subroutine (input1, input2, ... outfield);
```

Subroutine syntax varies for different FOCUS commands and phrases. These variations are discussed in subsequent sections.

Types of Arguments in Subroutine Calls

This section lists the acceptable arguments for each subroutine distributed in the subroutine library. Arguments are the values that you specify within the parentheses; this is also referred to as a “call list.” Arguments can take many different forms. They can be:

- Numeric constants, such as 6 or 15.
- Alphanumeric literals, such as 'STEVENS' or 'NEW YORK NY'. Literals are enclosed in single quotation marks.
- Numbers stored in alphanumeric format.
- Field names, such as FIRST_NAME or HIRE_DATE. Fields can be data source fields or temporary fields. The field names can be 66-character or qualified field names, unique truncations, or aliases.
- Expressions, such as numeric, date, and alphanumeric. Expressions can use the arithmetic operators and the concatenation sign (|). For example, an expression may consist of CURR_SAL * 1.03 or FN | | LN.

  For example, this calculation uses subroutine output:

  ```
  field= sub1(input1, input2,...'format') + sub2(input1, input2,...'format');
  ```

  The values returned by two subroutines are added and the result is stored in a field. The '*format*' argument in single quotation marks is the format of the value returned by each subroutine. The format argument is not supported in Maintain.
- Dialogue Manager variables, such as &CODE or &DDNAME.
- Date constants, such as '022894'.
- Formats of the output values, enclosed in single quotation marks.
- As input arguments for RECAP commands only, row and column references (R notation, E notation, or labels), or names of other RECAP calculations.

Rules for Arguments in Subroutine Calls

The following rules apply for arguments:

- Depending on the subroutine, arguments can be either alphanumeric or numeric:
    - Alphanumeric arguments (such as literals and alphanumeric fields) are stored internally as one character per byte. Numbers can also be stored in alphanumeric format. Literals are enclosed in single quotation marks, except when specified in operating system -RUN commands (-MVS RUN, for example).

      Note: If an argument is listed as having a specific alphanumeric format such as A8, it is a required format and you must specify it.
    - Numeric arguments (such as numeric constants and numeric fields) are stored internally as binary or packed numbers. This includes arguments in integer (I), floating-point (F), double-precision (D), and packed (P) formats.

      Note: If an argument is listed as having a numeric format, you may specify any of the four numeric formats (I, F, D, and P). If an argument is listed as having a specific numeric format such as double-precision, it is a required format and you must specify it.

  If you supply the wrong type of data for an argument, you will either cause an error or the subroutine will not return correct data.
- Arguments are passed to subroutines by reference, meaning that the memory location of the argument is passed. No indication of length of the argument is implied. The length, when needed (usually for alphanumeric strings) must be passed as another argument.

  When lengths of arguments are required, you must be careful to ensure that all lengths are correct. Some subroutines require a length for the input arguments and output arguments (for example, SUBSTR); others use one length for both input and output arguments (for example, UPCASE). In general, when one length is specified, it is used for both input and output fields.

  Providing an incorrect length can cause incorrect results:
    - If the specified length is shorter than the actual length, an initial subset of a string is used. For example, passing an argument of 'ABCDEF' and specifying a length of 3, is treated as a string of 'ABC'.
    - If the specified length is too long, whatever is in memory beyond the length is included. For example, passing an argument of 'ABC' and specifying a length of 6, is treated as a string beginning with 'ABC' plus whatever three characters are in the next 3 positions of memory. Depending on memory utilization, the extra three characters can be anything.

- Arguments must be specified in the exact order as shown for each subroutine; the order varies, according to each subroutine.
- The number of arguments varies, according to each subroutine. Subroutines may require up to six arguments.

  Customized subroutines may require any number of arguments. The maximum number of arguments per subroutine call is 28, including the output argument. If the subroutine requires more than 28 arguments, you must use two or more call statements to pass the arguments to the subroutine.
- Subroutine calls can serve as arguments in other subroutine calls or in FOCUS functions.
- You cannot specify a Dialogue Manager amper variable for the output argument without coding &VAR.EVAL. You may specify an amper variable as an input argument.
- Dialogue Manager converts arguments to double precision when it deems appropriate. The determination is made solely based on the value of the argument; not on what the subroutine expects for its pre-determined formats.

  If the argument is numeric (&arg.TYPE is 'N'), the value is converted to double precision. If the subroutine expects an alphanumeric string and the input is a numeric string, incorrect results will occur because of the conversion to double precision. To resolve this problem, append a non-numeric character to the end of the string, but do not count this extra character in the length of the argument.

  For example, to prevent the conversion of a delimiter blank character (' ') to a double precision zero in the GETTOK subroutine, include a non-numeric character after the blank (for example, ' @'). The GETTOK uses only the first character (the blank) as a delimiter and the extra character (@) prevents conversion to double precision.

Using Subroutine Calls in FOCUS Functions

Subroutines can serve as arguments in the FOCUS functions described in this chapter. For example, the MAX function returns the largest argument in a list. The statement

```
field = MAX (5000, subroutine (arguments, 'format'));
```

stores either the value 5000 or the value returned by the subroutine, whichever is larger, in a field.

Using Subroutine Calls in DEFINE, COMPUTE, and VALIDATE Commands

Subroutines may be called from the DEFINE command or Master File attribute, COMPUTE command, and VALIDATE command. The syntax is:

```
DEFINE [FILE filename]
tempfield[/format] = subroutine (input1, input2, input3, ...
{outfield|'format2'} );
```

```
COMPUTE
tempfield[/format] = subroutine (input1, input2, input3, ...
{outfield|'format2'} );
```

```
VALIDATE
tempfield[/format] = subroutine (input1, input2, input3, ...
{outfield|'format2'} );
```

The resulting temporary field is the same field that is specified for the outfield argument.

The temporary field’s format is required if it is the first time the field is defined; afterwards, it is optional.

If the subroutine returns output as the format of the output value (format2), the format of the temporary field must match the 'format2' argument. For example:

```
CENTER_NAME/A15=CTRFLD (LAST_NAME, 15, 'A15');
```

For a calculation or a compound IF statement, you must specify the format for the output value. There are two methods to do this:

- Pre-define the format of the output field with a separate statement. For example:

  ```
  COMPUTE
  AMOUNT/D8.2 =;
  AMOUNT_FLAG/A5 = IF subroutine(input1,input2,AMOUNT) GE 500
      THEN 'LARGE' ELSE 'SMALL';
  ```

  The AMOUNT field is pre-defined with the format D8.2. The subroutine returns a value to the output field AMOUNT (last argument). The IF statement tests if AMOUNT is greater or less than 500 and stores the result in the temporary flag AMOUNT_FLAG.

- Specify the last argument in the argument list as the format. For example:

  ```
  AMOUNT_FLAG/A5 = IF subroutine(input1,input2,'D8.2') GE 500
      THEN 'LARGE' ELSE 'SMALL';
  ```

  The statement tests the returned value directly. This is possible because the subroutine call defines the format of the return value (D8.2).

Using Subroutine Calls in WHERE and IF Tests

Subroutines may be specified in WHERE and IF selection tests. The output value of the subroutine is compared against the test value.

For example, this request prints employee names and current salaries for last names that begin with the letters MC. The SUBSTR subroutine extracts the first two characters as a substring.

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME LAST_NAME CURR_SAL
WHERE SUBSTR(15,LAST_NAME,1,2,2,'A15') IS 'MC';
END
```

The report returns as:

```
FIRST_NAME  LAST_NAME               CURR_SAL
----------  ---------               --------
JOHN        MCCOY                 $18,480.00
ROGER       MCKNIGHT              $16,100.00
```

Using Subroutine Calls in -SET Control Statements

In Dialogue Manager, -SET statements are used to create amper variables. To assign the returned value of a subroutine to an amper variable, use the syntax:

```
-SET &variable = subroutine (input1,&variable2[.LENGTH],...,'format');
```

The '*format*' argument is the format of the output value, enclosed in single quotation marks. You cannot specify a Dialogue Manager amper variable for the output argument ('*format*'); however, you may specify an amper variable as an input argument.

If a subroutine requires the length of a character string as an input argument, you may prompt for the character string, then use the suffix .LENGTH to test the length.

For example, this Dialogue Manager procedure prompts for a sentence (&SN), then uses the GETTOK subroutine, described on page 9-105, to extract the third word (token) from the sentence. The suffix .LENGTH passes the number of characters in the sentence to the subroutine. The extra character (%) is included to prevent the conversion of a delimiter blank character to a double precision zero (see *Rules for Arguments in Subroutine Calls* on page 9-15):

```
-PROMPT &SN.ENTER A SENTENCE.
-SET &WORD3 = GETTOK (&SN, &SN.LENGTH, 3, ' %', 30, 'A30');
```

Dialogue Manager variables only contain alphanumeric data. If a subroutine returns a numeric value and you set a Dialogue Manager variable to this value, FOCUS truncates and converts it to a character string before storing it in the variable based on the format.

Another example, the AYMD subroutine, described on page 9-41, adds 14 days to dates:

```
-SET &OUTDATE = AYMD (&INDATE, 14, 'I6');
```

The &INDATE variable for the input date is previously set in the procedure. The date is in the six-digit year-month-day format.

The format of the output date is a six-digit integer. Although the format ('I6') indicates that the output is an integer, it is stored in the &OUTDATE variable as a character string. For this reason, if you display the value of &OUTDATE, you will not see slashes separating the year, month, and day.

Using Subroutine Calls in -IF and IF Branching Statements

You can use subroutines in Dialogue Manager -IF and IF branching statements. The syntax is:

```
[-]IF subroutine (args) relation expression GOTO label1
[-]ELSE GOTO label2;
```

Specify input arguments and the format of the output ('*format*').

You may specify any valid relation and logical expression. Depending on whether the condition is true or false, the procedure branches to the specified Dialogue Manager label or MODIFY case.

For -IF statements:

- You cannot specify a Dialogue Manager amper variable for the output argument ('*format*') unless you use the &VAR.EVAL syntax; however, you may specify an amper variable as an input argument.
- If a subroutine requires the length of a character string as an input argument, you can prompt for the character string, then use the suffix .LENGTH to test for the length (see *Using Subroutine Calls in -SET Control Statements* on page 9-18).

This annotated example illustrates an -IF statement that executes one of two requests depending on when a planned project is expected to be completed.

```
    -LOOP
1.  -PROMPT &INDATE.ENTER START DATE IN YEAR-MONTH-DAY FORMAT OR ZERO TO EXIT:.
2.  -IF &INDATE EQ 0 GOTO EXIT;
3.  -SET &WEEKDAY = DOWK(&INDATE,'A4');
4.  -TYPE START DATE IS &WEEKDAY &INDATE
5.  -PROMPT &DAYS.ENTER ESTIMATED PROJECT LENGTH IN DAYS:.
6.  -IF AYMD(&INDATE,&DAYS,'I6YMD') LT 960101 GOTO EARLY;
    -TYPE LONG PROJECT
    -*EX LONGPROJ
7.  -RUN
    -GOTO LOOP
    -EARLY
    -TYPE SHORT PROJECT
    -*EX SHRTPROJ
8.  -RUN
    -GOTO LOOP
    -EXIT
```

This procedure processes as follows:

1. The procedure prompts you for a start date of a project in YYMMDD format.
2. If you enter a 0, the procedure terminates execution.
3. The DOWK subroutine obtains the day of week for the start date.
4. The -TYPE statement displays the day of week and date for the start of the project.
5. The procedure prompts you for the estimated length of the project in days.
6. The AYMD subroutine calculates the date that the project will finish. If this date is before January 1, 1996, the -IF statement branches to the label EARLY.
7. If the project will finish on or after January 1, 1996, the procedure types the words “LONG PROJECT” and returns to the top of the procedure.
8. If the procedure branches to the label -EARLY, it types the words “SHORT PROJECT” and returns to the top of the procedure.

Operating System -RUN Statements

You can call subroutines with all alphanumeric arguments from Dialogue Manager -CMS RUN, -TSO RUN, and -MVS RUN statements. These subroutines perform specific tasks but typically do not return any values (for instance, a private subroutine that clears the screen of a non-3270 terminal).

The syntax of the RUN statement is:

```
-CMS RUN subroutine, input1, input2, ... [,&output]
-TSO RUN subroutine, input1, input2, ... [,&output]
-MVS RUN subroutine, input1, input2, ... [,&output]
```

Separate the subroutine name and each argument with a comma. For alphanumeric literals used as arguments, do not enclose them in single quotation marks.

Specify an output argument as a Dialogue Manager variable if the subroutine returns a value; otherwise, omit it. If you specify an output variable, you must pre-define its length using a -SET statement. For example, if the subroutine requires an output argument that is eight bytes long, you need to define the variable with eight characters enclosed in single quotation marks before the subroutine call:

```
-SET &output = '12345678';
```

For subroutines that require arguments in numeric format, you must first convert the arguments (whether they are numeric constants or stored in variables) into double-precision numbers using the ATODBL subroutine, discussed on page 9-35. All numeric arguments in Dialogue Manager are stored in alphanumeric format and require conversion before being passed to subroutines. Unlike the -SET statement, operating system -RUN commands do not automatically convert them. You need to use the ATODBL subroutine, because the EDIT function cannot store double-precision numbers in Dialogue Manager variables.

If a subroutine requires the length of a character string as an input argument, you may prompt for the character string, then use the suffix .LENGTH to test the length. (See *Using Subroutine Calls in -SET Control Statements* on page 9-18 for an example.)

The following is an example of a subroutine that does not return any values. Suppose you write a subroutine called BLANKOUT that clears part of the screen on a Tektronix terminal (a non-3270 terminal). The subroutine reads one argument that indicates which part of the screen to blank out. To clear the top half of the screen, you include this statement in a procedure

```
-CMS RUN BLANKOUT, H1
```

or:

```
-TSO RUN BLANKOUT, H1
```

Using Subroutine Calls in WHEN Phrases

Subroutines may be called from the WHEN phrase as part of a Boolean expression. The syntax is

```
WHEN (value relation value) [{AND|OR} (value relation value)];
```

or:

```
WHEN NOT (logical expression)
```

Example Using a Subroutine Call in a WHEN Phrase

For example, this report request checks the values in the LAST_NAME field against a mask. It prints a subfoot message when the condition, a match, occurs. (**Note:** In this example, in order to produce a true condition, specify WHEN NOT, because the CHKFMT subroutine returns a 0 value when a match occurs.)

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT BY LAST_NAME
ON LAST_NAME SUBFOOT
"*** LAST NAME <LAST_NAME DOES MATCH MASK"
WHEN NOT CHKFMT (15, LAST_NAME,'SMITH          ','I6');
END
```

The report returns as:

```
LAST_NAME        DEPARTMENT
---------        ----------
BANNING          PRODUCTION
BLACKWOOD        MIS
CROSS            MIS
GREENSPAN        MIS
IRVING           PRODUCTION
JONES            MIS
MCCOY            MIS
MCKNIGHT         PRODUCTION
ROMANS           PRODUCTION
SMITH            MIS
                 PRODUCTION
*** LAST NAME SMITH DOES MATCH MASK
STEVENS          PRODUCTION
```

Using Subroutine Calls in RECAP Commands

Subroutines may be called from Extended Matrix Reporting (EMR) RECAP commands. The syntax is:

```
RECAP name[(n)][/format] = subroutine (input1,...,'format2');
```

Instead of a temporary field, specify the name of the calculation. You also may specify the number (*n*) of the column where you want the value displayed. If you omit the column number, the value appears in all columns.

The format of the calculation is optional; the default is D12.2. If the calculation consists of only the subroutine, make sure that the format of the subroutine output value ('*format2*') agrees with the calculation’s format. If the calculation format is larger than the column width, the value displays in that column as asterisks.

The input arguments for a RECAP command may include numeric constants, alphanumeric literals, row and column references (R notation, E notation, or labels) or names of other RECAP calculations.

Example Using a Subroutine in a RECAP Command

Suppose you have a subroutine named INVEST in your private collection of subroutines (INVEST is not available in the supplied library) and it calculates the amount on the basis of cash on hand, total assets, and the current date. In order to create a report that prints an account of company assets and calculates how much money the company has available to invest, you must create a report request that invokes the INVEST subroutine.

The current date is obtained from the &YMD system variable. The NOPRINT option beside it prevents the date from appearing in the report; the date is solely used as input for the next RECAP statement.

The request is:

```
TABLE FILE LEDGER
 HEADING CENTER
 "ASSETS AND MONEY AVAILABLE FOR INVESTMENT </2"
 SUM AMOUNT ACROSS HIGHEST YEAR
 IF YEAR EQ 1985 OR 1986
 FOR ACCOUNT
 1010 AS 'CASH'                                 LABEL CASH      OVER
 1020 AS 'ACCOUNTS RECEIVABLE'                  LABEL ACR       OVER
 1030 AS 'INTEREST RECEIVABLE'                  LABEL ACI       OVER
 1100 AS 'FUEL INVENTORY'                       LABEL FUEL      OVER
 1200 AS 'MATERIALS AND SUPPLIES'               LABEL MAT       OVER
 BAR                                                            OVER
 RECAP TOTCAS = CASH+ACR+ACI+FUEL+MAT; AS 'TOTAL ASSETS'        OVER
 BAR                                                            OVER
 RECAP THISDATE/A8 = &YMD; NOPRINT                              OVER
 RECAP INVAIL = INVEST(CASH,TOTCAS,THISDATE,'D12.2'); AS
           'AVAIL. FOR INVESTMENT'                              OVER
 BAR AS '='
 END
```

The request produces the following report:

```
PAGE     1

ASSETS AND MONEY AVAILABLE FOR INVESTMENT


                        YEAR
                          1986    1985
----------------------------------------
CASH                     2,100   1,684
ACCOUNTS RECEIVABLE        875     619
INTEREST RECEIVABLE      4,026   3,360
FUEL INVENTORY           6,250   5,295
MATERIALS AND SUPPLIES   9,076   7,754
                        ------  ------
TOTAL ASSETS            22,327  18,712
                        ------  ------
AVAIL. FOR INVESTMENT    3,481   2,994
                        ======  ======
```

Storing and Accessing External Subroutines

Accessing external subroutines varies by platform. The following topics describe how to access subroutines on specific platforms.

Note: If you have a private collection of subroutines (you created your own or use customized subroutines), do not store them in the subroutine library. Store them separately to avoid overwriting them whenever your site installs a new release.

Storing and Accessing Subroutines on MVS

In MVS, Information Builders-supplied subroutines are stored as part of FUSELIB.LOAD. In addition to this load library, your site may have private collections of subroutines stored in separate load libraries. Load libraries are partitioned data sets containing link-edited modules.

MVS Batch Allocation

To use subroutines stored as load libraries, allocate the load libraries to the ddname USERLIB in the FOCUS JCL or CLIST. For example, to allocate subroutines stored in BIGLIB.LOAD in JCL:

```
//USERLIB DD DISP=SHR,DSN=BIGLIB.LOAD
```

The FOCUS search order is USERLIB, STEPLIB, JOBLIB, link pack area, and linklist.

MVS/TSO Allocation

To use these subroutines in MVS/TSO, allocate the load libraries to ddname USERLIB. Issue the ALLOCATE command: 1) in MVS/TSO before going into your FOCUS session; or 2) from FOCUS before executing your request. You may also include the command in your PROFILE FOCEXEC.

The syntax is

```
{MVS|TSO} ALLOCATE FILE(USERLIB) DSN(lib1 lib2 lib3 ...) SHR
```

where:

`MVS or TSO`

Specify the prefix if you issue the ALLOCATE command from FOCUS or include it in your PROFILE FOCEXEC.

`lib1...`

Are the names of the load libraries. (This concatenates the data sets to ddname USERLIB.)

Note:

- If you have private collections of subroutines, you need to allocate those load libraries in addition to the FUSELIB load library.
- If you are in a FOCUS session, you may also use the DYNAM ALLOCATE command to specify the allocation.

For example, to allocate the FUSELIB.LOAD load library in a FOCUS session, use either the TSO ALLOCATE or DYNAM ALLOCATE command

```
TSO ALLOC F(USERLIB) DA('prefix.FUSELIB.LOAD') SHR
```

or

```
DYNAM ALLOC FILE USERLIB DA prefix.FUSELIB.LOAD SHR
```

where *prefix* is your high-level qualifier.

As another example, suppose a report request calls two subroutines: BENEFIT stored in library SUBLIB.LOAD, and EXCHANGE stored in library BIGLIB.LOAD. FOCUS can locate user-written subroutines in ddname USERLIB; therefore, the BIGLIB library is concatenated to USERLIB. Before executing the report request, enter:

```
DYNAM ALLOC FILE USERLIB DA SUBLIB.LOAD SHR
DYNAM ALLOC FILE BIGLIB  DA BIGLIB.LOAD SHR
DYNAM CONCAT FILE USERLIB BIGLIB
```

FOCUS searches the load libraries in the order that you specified them in the ALLOCATE command.

Or, for batch mode, concatenate the load library to the ddname STEPLIB or USERLIB in your JCL:

```
//FOCUS EXEC PGM=FOCUS
//STEPLIB     DD DSN=FOCUS.FOCLIB.LOAD,DISP=SHR
//            DD DSN=FOCUS.FUSELIB.LOAD,DISP=SHR
                            .
                            .
                            .
```

The FOCUS search order is: USERLIB, STEPLIB, JOBLIB, and link pack area and linklist.

Dynamic Language Environment Support

IBM's recommended platform for high-level language products is known as Language Environment (LE) for VM and MVS operating environments. It provides a unified platform for runtime services used by LE supported languages. FOCUS user-written subroutines can be linked using IBM's LE environment. It incorporates the automatic pre-initialization of a Language Environment Enclave or Sub-Enclave as required. LE support allows HLI subroutines with LE libraries to run most efficiently. Non-LE-linked subroutines 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.

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. You control use of IBM's Dynamic Language Environment by setting an option in FOCPARM, in a FOCUS application, or in your FOCUS session.

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.

Syntax How to Control the LE Environment Setting

The syntax for specifying the type of LE support you need is

```
SET IBMLE = support
```

Valid values are:

<u>`OFF`</u>

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 page 9-27.

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

Used only for user-created FORTRAN subroutines that need the LE environment. 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.

Note: The setting of ALL is not supported by IBM.

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 procedures 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	For performance reasons, link-edit the subroutine as reusable (reus). If the subroutine must be executed with IBMLE=OFF, apply the COBOL run-time option **rtereus** to the routine. Without this option, the subroutine returns invalid data and generates the following messages: IGZ0044S There was an attempt to call the COBOL main program [*xxxxxxxx*] that was not in initial state. The traceback information could not be determined. The job does not abend but produces a CEEDUMP prefaced by: CEE3DMP[*release*]: Condition processing resulted in the unhandled condition.
PL/I - LE	ON or ALL	If executed with IBMLE=OFF, results in a 0C1 abend.

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 procedure or FOCUS session are not supported and may produce unpredictable results.

Storing and Accessing Subroutines on VM/CMS

In CMS, Information Builders-supplied subroutines are stored as:

- The load library FUSELIB LOADLIB.
- The text library FUSELIB TXTLIB. A text library is a file that is composed of multiple text files called members. Subroutines can be stored as members of one or more text libraries. The file type for text libraries is TXTLIB.
- Text files. The file name of a text file must match the subroutine name. The file type is TEXT. For example, the EXCHANGE subroutine stored as a text file has the file identifier (ID):

  ```
  EXCHANGE TEXT
  ```

Note:

- In addition to the FUSELIB load library, your site may have private collections of subroutines stored in separate libraries or text files.
- If you create your own subroutines in text files or text libraries, the subroutine must be 31-bit addressable and created as part of a LOADLIB.

If your request calls subroutines stored as text files, FOCUS can find the subroutines automatically. Remember, though, that you must have access to the disks where the subroutines reside.

FOCUS searches for subroutines in the standard CMS search sequence:

1. Load libraries, in the order that you specified them in the GLOBAL LOADLIB command.
2. Text files, searching attached disks in alphabetical order.
3. Text libraries, in the order that you specified them in the GLOBAL TXTLIB command.

For subroutines stored as text files in CMS, the access method is automatic. When your request calls the subroutine, FOCUS searches attached disks in alphabetical order, provided that you have proper authorization.

For subroutines stored as load or text libraries in CMS, you need to issue the CMS GLOBAL command. The GLOBAL command enables FOCUS to search specified libraries for the subroutines.

Issue the command in CMS before starting the server. You may also include the command in your server’s global profile.

Note: Subroutines written in languages such as COBOL and PL/I, or subroutines that call system subroutines, require that the GLOBAL command also specify a system library. FUSELIB subroutines do not require any other system libraries.

Syntax How to Enable FOCUS to Search Specified Libraries for Subroutines

The syntax for the GLOBAL command is

```
[CMS] GLOBAL {LOADLIB|TXTLIB} library1 library2 library3 ...
```

where:

`CMS`

Specify this prefix if you issue the GLOBAL command from a profile or stored procedure, or include it in your server's global profile.

`library``1...`

Are the file names of the load and text libraries containing the subroutines. The maximum number of libraries is 63.

Note: If you have private collections of subroutines, you need to specify those libraries in the GLOBAL command in addition to the FUSELIB load library.

Syntax How to List Libraries Specified by the GLOBAL Command

To list load or text libraries specified by the GLOBAL command, issue:

```
CMS QUERY {LOADLIB|TXTLIB}
```

Example Using the GLOBAL COMMAND to Access Subroutines

For example, your server's global profile may contain the GLOBAL command:

```
CMS GLOBAL LOADLIB FUSELIB
```

For another example, suppose a report request calls two subroutines: BENEFIT, stored in text library SUBLIB, and EXCHANGE, stored in text library BIGLIB. Before executing the request, issue the GLOBAL command in a stored procedure or at the command line:

```
CMS GLOBAL TXTLIB SUBLIB BIGLIB
```

If you issue two GLOBAL commands, the second command replaces the first. Once a library is opened (as a result of referencing one of its members), the library cannot be changed until you exit FOCUS.

Alphabetical List of Functions and Subroutines

The following sections describe the functions and subroutines in alphabetical order.

ABS: Calculating Absolute Value

The ABS function returns the absolute value of its argument.

Available on: All platforms.

Syntax How to Calculate Absolute Value

The syntax is

```
ABS(argument)
```

where:

argument

Numeric

Is the value on which ABS operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

Example Report Request Calculating Absolute Value of Difference Between UNIT_SOLD and DELIVER_AMT

The following example calculates the absolute value of the difference between the number of units sold and the number delivered:

```
TABLE FILE SALES
PRINT UNIT_SOLD AND DELIVER_AMT AND
COMPUTE DIFF/I5 = DELIVER_AMT - UNIT_SOLD; AND
COMPUTE ABS_DIFF/I5 = ABS(DIFF);
BY PROD_CODE
WHERE DATE LE '1017';
END
```

The example produces the following report:

```
PAGE     1


PROD_CODE  UNIT_SOLD  DELIVER_AMT  DIFF  ABS_DIFF
---------  ---------  -----------  ----  --------
B10               30           30     0         0
B17               20           40    20        20
B20               15           30    15        15
C17               12           10    -2         2
D12               20           30    10        10
E1                30           25    -5         5
E3                35           25   -10        10
```

ARGLEN: Measuring Argument Length

The ARGLEN subroutine measures the argument length of a character string within a field, excluding trailing spaces. (The field format specifies the length of the field, including trailing spaces.)

Note that in Dialogue Manager, you can measure the length of prompted character strings using the .LENGTH suffix.

Available on: All platforms.

Syntax How to Measure the Length of a Character String

The syntax is

```
ARGLEN(inlength, infield, outfield)
```

where:

inlength

Integer

Is the total length of the field containing the character string.

infield

Alphanumeric

Is the name of the field for which the argument length is to be determined.

outfield

Integer

Is the field to which the integer result is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Measuring Length of Employee Last Names

The following example displays the lengths of employee last names:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
NAME_LEN/I3 = ARGLEN(15, LAST_NAME, NAME_LEN);
WHERE DEPARTMENT EQ 'MIS';
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        NAME_LEN
---------        --------
SMITH                   5
JONES                   5
MCCOY                   5
BLACKWOOD               9
GREENSPAN               9
CROSS                   5
```

ASIS: Distinguishing Between a Blank and a Zero

The ASIS function is used in Dialogue Manager to distinguish between a blank and a zero. By using the ASIS function in Dialogue Manager, numeric string constants and variables defined as numeric strings (numerics within single quotation marks) can be differentiated from fields defined simply as numeric. The ASIS function forces FOCUS to evaluate a variable as it is entered rather than converting it to a number. It is used in Dialogue Manager equality expressions only.

Available on: MVS, VM/CMS, WebFOCUS.

Syntax How to Distinguish Between a Blank and a Zero

The syntax is

```
ASIS(argument)
```

where:

argument

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. The expression may call a function or a subroutine.

If you specify an alphanumeric literal, enclose it in single quotation marks (’). If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

Example Distinguishing Between a Blank and a Zero

The following examples show how the ASIS function affects the way FOCUS recognizes values. In the first example, the ASIS function is not used. FOCUS does not distinguish between variables defined as ' ' and 0:

```
-SET &VAR1 = ' ';
-SET &VAR2 = 0;
-IF &VAR2 EQ &VAR1 GOTO ONE;
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 NOT TRUE
-QUIT
-ONE
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 TRUE
```

The output is:

```
VAR1  EQ VAR2 0 TRUE
```

The second example shows the use of the ASIS function to distinguish between the two variables:

```
-SET &VAR1 = ' ';
-SET &VAR2 = 0;
-IF &VAR2 EQ ASIS(&VAR1) GOTO ONE;
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 NOT TRUE
-QUIT
-ONE
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 TRUE
```

The output is:

```
VAR1  EQ VAR2 0 NOT TRUE
```

ATODBL: Converting Alphanumeric Strings to a Double-Precision Number

The ATODBL subroutine converts numbers from alphanumeric to double-precision format. You use this subroutine primarily to prepare arguments for other subroutines that you call from Dialogue Manager -CMS RUN, -TSO RUN, and -MVS RUN statements. For other applications, the EDIT function performs the same operation.

All numeric arguments in Dialogue Manager are in alphanumeric format. These arguments must be converted to double-precision format before being passed to subroutines. The -SET statements automatically convert these arguments, but -CMS RUN, -TSO RUN, and -MVS RUN statements do not.

In order to call a subroutine from an operating system -RUN statement, you must convert each numeric argument into double-precision format and store it in a Dialogue Manager variable. The variable is used in the subroutine argument list. Since the EDIT function cannot store double-precision numbers in Dialogue Manager variables, you must call the ATODBL subroutine to convert the arguments.

Available on: All platforms.

Related functions and subroutines:

- EDIT
- FTOA

Two syntaxes for the ATODBL subroutine exist.

Procedure How to Convert Alphanumeric Strings to a Double-Precision Number From an Operating System -RUN Statement

To use the ATODBL subroutine in Dialogue Manager, perform these steps:

1. Define the output variable as 8 bytes long. The syntax is

```
-SET &outfield = '12345678';
```

where:

&outfield

Is the output variable. The value must be eight characters, enclosed in single quotation marks.

2. Call the ATODBL subroutine from an operating system -RUN statement, not from a -SET statement. The syntax is

```
-{operating_system} RUN ATODBL, number, inlength, &outfield
```

where:

operating_system

Is CMS, TSO, or MVS.

number

Alphanumeric

Is the number you want to convert or the variable containing the number. The number can be up to 15 bytes long. It can contain a sign and a decimal point but no other character; otherwise, the subroutine returns a 0.

inlength

Alphanumeric

Is the number of bytes in the *number* argument; maximum value is 15.

Note: This must be a character string.

outfield

A8

Is the predefined output variable.

Syntax How to Convert Alphanumeric Strings to a Double-Precision Number From a Non-Dialogue Manager Statement

The syntax for specifying the ATODBL subroutine in other applications (except from -SET statements) is

```
ATODBL(number, inlength, outfield)
```

where:

number

Alphanumeric

Is the number to be converted, the field that contains the number, or a variable.

inlength

Alphanumeric

Is the number of bytes in the *number* argument; maximum value is 15. If you are specifying this field as a numeric constant, enclose it in single quotation marks.

outfield

Double-Precision

Is the name of the field that contains the double-precision number. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

Example MODIFY Request Converting a Prompted Alphanumeric Value Into a Double-Precision Number

For the following example, the MISSING attribute is specified for the CURR_SAL field in the EMPLOYEE Master File. This means that, if you do not enter a current salary value for this double-precision field, the null is interpreted as a default value, a period.

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,       ALIAS=EID,     FORMAT=A9,       $
    .
    .
    .
 FIELDNAME=CURR_SAL,     ALIAS=CSAL,FORMAT=D12.2M, MISSING=ON,$
    .
    .
    .
```

In this MODIFY procedure, the ATODBL subroutine converts the alphanumeric value from the TCSAL field to double-precision format. After you enter an employee ID and the employee's last and first names display, you are prompted to supply a current salary or the characters N/A, if one is not available. The current salary value is stored in a temporary alphanumeric field, TCSAL. The ATODBL converts the alphanumeric value and the TYPE statement displays it.

```
MODIFY FILE EMPLOYEE
COMPUTE TCSAL/A12=;
PROMPT EID
MATCH EID
ON NOMATCH REJECT
ON MATCH TYPE "EMPLOYEE <D.LAST_NAME <D.FIRST_NAME"
ON MATCH TYPE "ENTER CURRENT SALARY OR 'N/A' IF NOT AVAILABLE"
ON MATCH PROMPT TCSAL
ON MATCH COMPUTE
CSAL MISSING ON=IF TCSAL EQ 'N/A' THEN MISSING
                ELSE ATODBL(TCSAL,'12','D12.2');
ON MATCH TYPE "SALARY NOW <CSAL"
DATA
```

A sample execution is as follows:

```
 EMPLOYEEFOCUS   A ON 11/14/96 AT 13.42.55
 DATA FOR TRANSACTION    1

 EMP_ID      =
071382660
 EMPLOYEE STEVENS ALFRED
 ENTER CURRENT SALARY OR 'N/A' IF NOT AVAILABLE
 TCSAL       =
n/a
 SALARY NOW
 DATA FOR TRANSACTION    2

 EMP_ID      =
112847612
 EMPLOYEE SMITH MARY
 ENTER CURRENT SALARY OR 'N/A' IF NOT AVAILABLE
 TCSAL       =
45000
 SALARY NOW      $45,000.00
 DATA FOR TRANSACTION    3

 EMP_ID      =
end
 TRANSACTIONS:         TOTAL =     2  ACCEPTED=     2  REJECTED=     0
 SEGMENTS:             INPUT =     0  UPDATED =     0  DELETED =     0
```

The procedure processes as:

1. For the first transaction, the procedure prompts you for an employee ID. You enter: 071382660.
2. The procedure displays the last and first name of the employee, STEVENS ALFRED.
3. Then it prompts you for a current salary. You enter: N/A.
4. It displays a period (.).
5. For the second transaction, the procedure prompts you for an employee ID. You enter: 112847612.
6. The procedure displays the last and first name of the employee, SMITH MARY.
7. Then it prompts you for a current salary. You enter: 45000.
8. It displays $45,000.00.

AYM: Adding or Subtracting Months to or From Dates

The AYM subroutine adds and subtracts months from dates. The dates must be in year-month format. You can convert a date to this format by using the CHGDAT subroutine or the EDIT function.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

Available on: All platforms.

Related subroutines:

- CHGDAT
- EDIT
- AYMD

Syntax How to Add or Subtract Months to or From Dates

The syntax is

```
AYM(indate, months, outfield)
```

where:

indate

Numeric

Is the input date in year-month format. If the date is not valid, AYM returns a 0.

months

Integer

Is the number of months you are adding or subtracting from the date. To subtract months, make the number negative.

outfield

Integer

Is the name of the field to which the resulting date in year-month format is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Tip: If the input date is in integer year-month-day format (I6YMD or I8YYMD), simply divide the date by 100 to convert to year-month format and set the result to be an integer. This causes the day portion of the date, which is now after the decimal point, to be dropped.

Example Report Request Adding Six Months to Hire Date

The following example adds six months to the hire dates of employees. Note that the Compute expression converts the dates from year-month-day to year-month formats by dividing the dates by 100.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
HIRE_MONTH/I4YM = HIRE_DATE/100 ;
AFTER6MONTHS/I4YM = AYM(HIRE_MONTH, 6, AFTER6MONTHS);
BY LAST_NAME BY FIRST_NAME
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME  HIRE_DATE  HIRE_MONTH  AFTER6MONTHS
---------        ----------  ---------  ----------  ------------
BANNING          JOHN         82/08/01       82/08         83/02
BLACKWOOD        ROSEMARIE    82/04/01       82/04         82/10
CROSS            BARBARA      81/11/02       81/11         82/05
GREENSPAN        MARY         82/04/01       82/04         82/10
IRVING           JOAN         82/01/04       82/01         82/07
JONES            DIANE        82/05/01       82/05         82/11
MCCOY            JOHN         81/07/01       81/07         82/01
MCKNIGHT         ROGER        82/02/02       82/02         82/08
ROMANS           ANTHONY      82/07/01       82/07         83/01
SMITH            MARY         81/07/01       81/07         82/01
                 RICHARD      82/01/04       82/01         82/07
STEVENS          ALFRED       80/06/02       80/06         80/12
```

AYMD: Adding or Subtracting Days to or From Dates

The AYMD subroutine takes a valid date in the form [YY]YYMMDD and adds or subtracts a given number of days from the submitted date.

AYMD only operates on dates in year-month-day format. You can convert a date to this format using the CHGDAT subroutine or the EDIT function.

If the addition (or subtraction) of days crosses forward or back into a century, the century digits of the output year are adjusted.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

Available on: All platforms.

Related subroutines:

- CHGDAT
- EDIT
- AYM

Syntax How to Add or Subtract Days to or From Dates

The syntax is

```
AYMD(indate, days, outfield)
```

where:

indate

Integer

Is the input date in [YY]YYMMDD format. If the date is not valid, the subroutine returns a 0. If *indate* is a field name, it must refer to a field with I6, I6YMD, I8, I8YYMD, P6, P6YMD, F6, F6YMD, D6, or D6YMD format.

days

Integer

Is the number of days you are adding to *indate*. To subtract days, make the number negative.

outfield

I6, I6YMD, I8, or I8YYMD

Is the name of the field to which the resulting date is returned. This argument can also be the format of the output value, enclosed in single quotation marks. If *indate* is a field, both fields must have the same format.

Example Report Request Adding 35 Days to Hire Date

The following example adds 35 days to the hire date of employees:

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
AFTER35DAYS/I6YMD = AYMD(HIRE_DATE, 35, AFTER35DAYS);
BY LAST_NAME BY FIRST_NAME
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME  HIRE_DATE  AFTER35DAYS
---------        ----------  ---------  -----------
BANNING          JOHN         82/08/01     82/09/05
BLACKWOOD        ROSEMARIE    82/04/01     82/05/06
CROSS            BARBARA      81/11/02     81/12/07
GREENSPAN        MARY         82/04/01     82/05/06
IRVING           JOAN         82/01/04     82/02/08
JONES            DIANE        82/05/01     82/06/05
MCCOY            JOHN         81/07/01     81/08/05
MCKNIGHT         ROGER        82/02/02     82/03/09
ROMANS           ANTHONY      82/07/01     82/08/05
SMITH            MARY         81/07/01     81/08/05
                 RICHARD      82/01/04     82/02/08
STEVENS          ALFRED       80/06/02     80/07/07
```

BAR: Producing Bar Charts

The BAR subroutine enables you to produce horizontal bar charts in reports. The bars, which consist of repeating characters, constitute a field with each bar as a field value. When the report request prints the field, the bars appear in the report.

Available on: MVS, VM/CMS, OpenVMS.

Syntax How to Produce Bar Charts

The syntax is

```
BAR(barlength, infield, maxvalue, 'char', outfield)
```

where:

barlength

Numeric

Is the maximum length of the bar in repeating characters. If this value is less than or equal to 0, the subroutine does not return a bar.

infield

Numeric

Is the field you wish to illustrate as a bar chart.

maxvalue

Numeric

Is the maximum length of a bar. This value should be greater than the maximum value stored in the input field (*infield*). If an *infield* value is larger than the *maxvalue* argument, the subroutine uses *maxvalue* and returns a bar at maximum length.

'char'

Alphanumeric

Is the repeating character that creates the bars. If the argument specifies more than one character, only the first character is used to create the bars.

outfield

Alphanumeric

Is the name of the field that contains the bars. This output field must be large enough to contain a bar at maximum length, as defined by the *barlength* argument. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

Example Creating a Bar Chart of CURR_SAL

The following request prints the salaries of employees and graphs them with a bar chart. The maximum length of a bar is 30 characters long. The 30-character bar represents a maximum salary of $30,000 (which is $29,700, rounded up). Each equal sign represents approximately $1,000. The request is:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
    SAL_BAR/A30 = BAR(30, CURR_SAL, 30000, '=', SAL_BAR);
BY LAST_NAME BY FIRST_NAME
END
```

The request produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME           CURR_SAL  SAL_BAR
---------        ----------           --------  -------
BANNING          JOHN               $29,700.00  ==============================
BLACKWOOD        ROSEMARIE          $21,780.00  ======================
CROSS            BARBARA            $27,062.00  ===========================
GREENSPAN        MARY                $9,000.00  =========
IRVING           JOAN               $26,862.00  ===========================
JONES            DIANE              $18,480.00  ==================
MCCOY            JOHN               $18,480.00  ==================
MCKNIGHT         ROGER              $16,100.00  ================
ROMANS           ANTHONY            $21,120.00  =====================
SMITH            MARY               $13,200.00  =============
                 RICHARD             $9,500.00  =========
STEVENS          ALFRED             $11,000.00  ===========
```

Example Creating a Bar Chart of CURR_SAL With Scale

You may find it useful to print a scale over the bar chart. Consider this request which replaces the name of the computed field with a scale:

```
TABLE FILE EMPLOYEE
HEADING
"CURRENT SALARIES OF EMPLOYEES"
"GRAPHED IN THOUSANDS OF DOLLARS"
" "
PRINT CURR_SAL AS 'CURRENT_SALARY'
AND COMPUTE
    SAL_BAR/A30 = BAR(30, CURR_SAL, 30000, '=', SAL_BAR);
    AS
'    5   10   15   20   25   30,----+----+----+----+----+----+'
BY LAST_NAME AS 'LAST_NAME'
BY FIRST_NAME AS 'FIRST_NAME'
END
```

The request produces the following report:

```
PAGE     1

CURRENT SALARIES OF EMPLOYEES
GRAPHED IN THOUSANDS OF DOLLARS
                                                  5   10   15   20   25   30
LAST_NAME        FIRST_NAME   CURRENT_SALARY  ----+----+----+----+----+----+
---------        ----------   --------------  ------------------------------
BANNING          JOHN              $29,700.00  ==============================
BLACKWOOD        ROSEMARIE         $21,780.00  ======================
CROSS            BARBARA           $27,062.00  ===========================
GREENSPAN        MARY               $9,000.00  =========
IRVING           JOAN              $26,862.00  ===========================
JONES            DIANE             $18,480.00  ==================
MCCOY            JOHN              $18,480.00  ==================
MCKNIGHT         ROGER             $16,100.00  ================
ROMANS           ANTHONY           $21,120.00  =====================
SMITH            MARY              $13,200.00  =============
                 RICHARD            $9,500.00  =========
STEVENS          ALFRED            $11,000.00  ===========
```

BITSON: Determining if Bits Are On or Off

The BITSON subroutine evaluates individual bits within character strings to determine if a bit is on or off. If the bit is on, the subroutine returns a value of 1; otherwise, it returns a value of 0. This subroutine is useful in interpreting multi-punch data, where each punch conveys an item of information.

Available on: MVS, VM/CMS, UNIX, OpenVMS, WebFOCUS.

Syntax How to Determine if Bits Are On or Off

The syntax is

```
BITSON(bitnumber, infield, outfield)
```

where:

bitnumber

Integer

Is the number of the bit to be evaluated, counting from the left-most bit in the character string (counting from 1).

infield

Alphanumeric

Is the character string, enclosed in single quotation marks, or the field that contains the character string. The character string is in multiple 8-bit blocks.

outfield

Integer

Is the name of the field that contains the value of the bit: 1 or 0. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Evaluating the Twenty-Fourth Bit of LAST_NAME

This report request evaluates the twenty-fourth bit of the names stored in the LAST_NAME field.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
   BIT_24/I1 = BITSON(24, LAST_NAME, BIT_24);
END
```

The request produces the following report:

```
PAGE     1


LAST_NAME        BIT_24
---------        ------
STEVENS               1
SMITH                 1
JONES                 1
SMITH                 1
BANNING               1
IRVING                1
ROMANS                0
MCCOY                 1
BLACKWOOD             1
MCKNIGHT              0
GREENSPAN             1
CROSS                 0
```

BITVAL: Evaluating Bit Strings as Binary Integers

The BITVAL subroutine evaluates strings of bits within character strings. The bit strings can be any group of bits within the character string and can cross byte and word boundaries. The subroutine evaluates the bit strings as binary integers and returns the corresponding values.

Available on: MVS, VM/CMS, UNIX, OpenVMS, WebFOCUS.

Example How to Evaluate Bit Strings

The syntax is

```
BITVAL(infield, startbit, number, outfield)
```

where:

infield

Alphanumeric

Is the character string or field that contains the bit string.

startbit

Integer

Is the number of the first bit in the bit string, counting from the left-most bit in the character string. If this argument is less than or equal to 0, the subroutine returns a value of zero (0).

number

Integer

Is the number of bits in the bit string. If this argument is less than or equal to 0, the subroutine returns a value of zero (0).

outfield

Integer

Is the name of the field that contains the integer equivalent. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Evaluating Bits 12 Through 20 of LAST_NAME

This report request evaluates bits 12 through 20 of the last names stored in the field LAST_NAME:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
   STRING_VAL/I5 = BITVAL(LAST_NAME, 12, 9, 'I5');
END
```

The resulting report is:

```
PAGE     1


LAST_NAME        STRING_VAL
---------        ----------
STEVENS                  60
SMITH                   332
JONES                   365
SMITH                   332
BANNING                  29
IRVING                  414
ROMANS                  365
MCCOY                    60
BLACKWOOD               316
MCKNIGHT                 61
GREENSPAN               412
CROSS                   413
```

BYTVAL: Translating a Character to Its ASCII or EBCDIC Code

The BYTVAL subroutine translates characters to the ASCII or EBCDIC decimal values that represent them.

Available on: All platforms.

Syntax How to Translate a Character

The syntax is

BYTVAL(*character*, *outfield*)

where:

character

Alphanumeric

Is the input character. If you supply more than one character in this argument, the subroutine evaluates the first character. You can specify a field or amper variable that contains the character, or specify the character itself.

`outfield`

Integer

Is the name of the field to which the corresponding decimal value—an integer between 0 and 255—is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Dialogue Manager Procedure Returning EBCDIC Value for Prompted Character

This Dialogue Manager procedure prompts for a character, then returns the corresponding number:

```
-PROMPT &CHAR.ENTER THE CHARACTER TO BE DECODED.
-SET &CODE = BYTVAL (&CHAR, 'I3');
-TYPE
-TYPE THE EQUIVALENT VALUE IS &CODE
```

Suppose you want to know the equivalent value of the exclamation point (!). A sample execution is:

```
ENTER THE CHARACTER TO BE DECODED
!

THE EQUIVALENT VALUE IS 90
>
```

The procedure processes as:

1. When you execute the procedure, it prompts:

   ```
   ENTER THE CHARACTER TO BE DECODED
   ```

2. You enter an exclamation point: !.

3. The procedure responds:

   ```
   THE EQUIVALENT VALUE IS 90
   ```

CHGDAT: Changing Date Formats

The CHGDAT subroutine rearranges the year, month, and day portions of dates and converts dates between long and short date formats. Long formats contain the year, month, and day; short formats contain one or two of these elements, such as year and month or just day. A format can be longer if four digits are used for the year (for example, 1987), or shorter if only the last two digits are used (for example, 87).

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

Available on: All platforms.

Related functions and subroutines:

- DA subroutines
- DT subroutines

Syntax How to Change Date Formats

The syntax is

```
CHGDAT('oldformat', 'newformat', indate, outfield)
```

where:

`'oldformat'`

A5

Is the format of the input date.

`'newformat'`

A5

Is the format of the converted date.

indate

Alphanumeric

Is the input date. If the date is in numeric format, change it to alphanumeric format using the EDIT function. If the input date is invalid, the subroutine returns spaces.

outfield

Alphanumeric or A17

Is the name of the field to which the converted date is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Tip: Since CHGDAT returns the date in alphanumeric format with 17 characters, you can use the EDIT function to truncate this field to a shorter field or to convert the date to numeric format.

The date formats specified by the arguments *oldformat* and *newformat* contain the following characters in any combination:

`D` Days in the month (01 through 31).

`M` Months in the year (01 through 12).

`Y[Y]` Year. One Y indicates a two-digit date (such as 94); two Y's indicate a four-digit date (such as 1994).

If you want to spell out the month rather than use a number for the month, you can append one of the following to the format of the new date:

`T` Displays the month as a three-letter English abbreviation.

`X` Displays the full English name of month.

Any other character in the format is ignored.

If you are converting a date from short to long format (for example, from year-month to year-month-day), the subroutine supplies the portion of the date missing in the short format, as shown in the following table:

Portion of Date Missing	**Portion Supplied by the Subroutine**
Day (that is, from YM to YMD)	Last day of the month.
Month (that is, from Y to YM)	The month 12 (December).
Converting year from short to long form (that is, from YMD to YYMD)	The century will be determined by the 100-year window defined by DEFCENT and YRTHRESH. See the *Developing Applications* manual for details on DEFCENT and YRTHRESH.

Example Report Request Converting Numeric Date to Full Name

The following example displays the names and hire dates of employees, both in year-month-day format and in month-day-year format. The second format displays the full name of the month and the full year.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
ALPHA_HIRE/A17 = EDIT(HIRE_DATE); NOPRINT AND COMPUTE
HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX', ALPHA_HIRE, 'A17');
BY LAST_NAME BY FIRST_NAME
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME  HIRE_DATE  HIRE_MDY
---------        ----------  ---------  --------
BANNING          JOHN         82/08/01  AUGUST 01 1982
BLACKWOOD        ROSEMARIE    82/04/01  APRIL 01 1982
CROSS            BARBARA      81/11/02  NOVEMBER 02 1981
GREENSPAN        MARY         82/04/01  APRIL 01 1982
IRVING           JOAN         82/01/04  JANUARY 04 1982
JONES            DIANE        82/05/01  MAY 01 1982
MCCOY            JOHN         81/07/01  JULY 01 1981
MCKNIGHT         ROGER        82/02/02  FEBRUARY 02 1982
ROMANS           ANTHONY      82/07/01  JULY 01 1982
SMITH            MARY         81/07/01  JULY 01 1981
                 RICHARD      82/01/04  JANUARY 04 1982
STEVENS          ALFRED       80/06/02  JUNE 02 1980
```

CHKFMT: Checking String Format

The CHKFMT subroutine checks character strings for incorrect character types. It compares each string to a second string called a “mask,” comparing each character in the input string to the corresponding character in the mask. If all characters in the string match the characters or character types of those in the mask, CHKFMT returns the value 0. Otherwise, CHKFMT returns a value equal to the position of the first character in the string not matching the mask.

Available on: All platforms.

Syntax How to Check String Format

The syntax is

CHKFMT(*numchar*, *infield*, '*mask*', *outfield*)

where:

numchar

Integer

Is the number of characters you want to compare against the mask.

infield

Alphanumeric

Is the character string you are inspecting (enclosed in single quotation marks) or the field containing the string.

'*mask*'

Alphanumeric

Is the mask as described with the character symbols (enclosed in single quotation marks).

outfield

Integer

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

Some characters in the mask are generic: they represent character types. If a character in the string is compared to one of these characters and is the same type, it matches. These generic characters are:

A Represents any of the letters A-Z (uppercase or lowercase).

9 Represents any of the digits 0-9.

X Represents any of the letters A-Z or digits 0-9.

$ Represents any character.

Any other character in the mask represents only that character. For example, if the third character in the mask is the letter B, the third character in the string must be the letter B to match.

If the mask is shorter than the character string, the subroutine checks only the portion of the character string corresponding to the mask. For example, if you are using a four-character mask to test a nine-character string, only the first four characters in the string are checked; the rest are returned as a nomatch with CHKFMT giving the position as a result.

Example Report Request Checking the Format of EMP_ID

The following example checks the format of EMP_ID against a mask for nine numeric characters, beginning with the numerals 11.

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND LAST_NAME AND
COMPUTE CHK_ID/I3 = CHKFMT(9, EMP_ID, '119999999', CHK_ID);
END
```

The example produces the following report:

```
PAGE     1


EMP_ID     LAST_NAME     CHK_ID
------     ---------     ------
071382660  STEVENS            1
112847612  SMITH              0
117593129  JONES              0
119265415  SMITH              0
119329144  BANNING            0
123764317  IRVING             2
126724188  ROMANS             2
219984371  MCCOY              1
326179357  BLACKWOOD          1
451123478  MCKNIGHT           1
543729165  GREENSPAN          1
818692173  CROSS              1
```

Example MODIFY Request Checking the Format of EMP_ID

The following MODIFY procedure adds records of new employees to the EMPLOYEE data source. Each transaction begins as an employee ID that is alphanumeric with the first five characters as digits. The procedure rejects records with other characters in the employee ID. The procedure is:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID LAST_NAME FIRST_NAME DEPARTMENT
MATCH EMP_ID
    ON MATCH REJECT
    ON NOMATCH COMPUTE
        BAD_CHAR/I3 = CHKFMT (5, EMP_ID, '99999', BAD_CHAR);
    ON NOMATCH VALIDATE
        ID_TEST = IF BAD_CHAR EQ 0 THEN 1 ELSE 0;
        ON INVALID TYPE
            "BAD EMPLOYEE ID: <EMP_ID"
            "INVALID CHARACTER IN POSITION <BAD_CHAR"
    ON NOMATCH INCLUDE
    LOG INVALID MSG OFF
DATA
```

A sample execution is:

```
>
 EMPLOYEEFOCUS   A ON 12/05/96 AT 15.42.03
 DATA FOR TRANSACTION    1

 EMP_ID      =
111w2
 LAST_NAME   =
johnson
 FIRST_NAME  =
greg
 DEPARTMENT  =
production
 BAD EMPLOYEE ID: 111W2
 INVALID CHARACTER IN POSITION   4
 DATA FOR TRANSACTION    2

 EMP_ID      =
end
 TRANSACTIONS:         TOTAL =     1  ACCEPTED=     0  REJECTED=     1
 SEGMENTS:             INPUT =     0  UPDATED =     0  DELETED =     0
>
```

The procedure processes as:

1. The procedure prompts you for an employee ID, last name, first name, and department assignment. You enter the following data:

   ```
   EMP_ID:            111w2
   LAST_NAME:         johnson
   FIRST_NAME:        greg
   DEPARTMENT:        production
   ```

2. The procedure searches the data source for the ID 111W2. If it does not find this ID, it continues processing the transaction.
3. The CHKFMT subroutine checks the ID against the mask 99999, which represents five digits.
4. The fourth character in the ID, the letter “W”, is not a digit. The subroutine returns the value 4 to the BAD_CHAR field.
5. The VALIDATE statement tests the BAD_CHAR field. Since BAD_CHAR is not equal to 0, the procedure rejects the transaction and displays a message indicating the position of the invalid character in the ID.

CHKPCK: Validating Packed Fields

The CHKPCK subroutine validates packed fields, checking that their values are in packed format. The subroutine prevents data exceptions that occur when requests read packed fields from files containing values that are not valid packed numbers.

Available on: MVS, VM/CMS, UNIX, OpenVMS, WebFOCUS.

Syntax How to Validate Packed Fields

The syntax is

```
CHKPCK(inlength, infield, error, outfield)
```

where:

inlength

Numeric

Is the field length to be validated, from 1 to 16 bytes.

infield

Alphanumeric

Is the input field to be validated. The field is described as alphanumeric, not packed.

error

Numeric

Is the error code that the subroutine returns if a value is not packed. The error code is first truncated to an integer, then converted to packed. (The error code may appear on a report with a decimal point because of the format of the output field.) Choose an error code outside the range of data.

outfield

Packed

Is the name of the field that contains the input value if the value is packed or the error code. This argument can also be the format of the output value, enclosed in single quotation marks.

To use the CHKPCK subroutine, use these steps:

1. Make sure that the Master File (FORMAT, USAGE, and ACTUAL attributes), or the MODIFY FIXFORM statement describing the file, defines the field as alphanumeric, not packed. This does *not* change the field data, which remains packed. It enables the request to read the data without causing data exceptions.

2. Call the CHKPCK subroutine to examine the field. The subroutine returns its output to a field defined as packed. If the value it examines is a valid packed number, the subroutine returns the value; otherwise, it returns an error code.

Example

Validating Packed Data

In order to reproduce this example, you need to prepare a data source with invalid packed data. Issue this request:

```
DEFINE FILE EMPLOYEE
PACK_SAL/A8 = IF EMP_ID CONTAINS '123'
      THEN 'AAA' ELSE PCKOUT(CURR_SAL, 8, 'A8');
END
```

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT PACK_SAL BY EMP_ID
ON TABLE SAVE AS TESTPACK
END
```

The request creates the TESTPACK file that is used in this example. The file contains employee IDs, department assignments, and salaries. The salary field named PACK_SAL is defined as alphanumeric but contains packed data. The invalid packed data is a string of three letters (AAA).

```
>
 NUMBER OF RECORDS IN TABLE=       12  LINES=     12


 EBCDIC  RECORD NAMED  TESTPACK
 FIELDNAME                         ALIAS         FORMAT        LENGTH

 EMP_ID                            EID           A9              9
 DEPARTMENT                        DPT           A10            10
 PACK_SAL                                        A8              8

TOTAL                                                          27
DCB USED WITH FILE TESTPACK IS DCB=(RECFM=FB,LRECL=00027,BLKSIZE=00540)
>
```

Next, you must create a Master File for the TESTPACK file. If the description defines the salary field as packed, the bad values will cause data exceptions when a request reads the file. Instead, define the field as alphanumeric both in the USAGE and ACTUAL attributes:

```
FILE  = TESTPACK,  SUFFIX = FIX
FIELD = EMP_ID    ,ALIAS = EID,FORMAT = A9 ,ACTUAL = A9 ,$
FIELD = DEPARTMENT,ALIAS = DPT,FORMAT = A10,ACTUAL = A10,$
FIELD = PACK_SAL  ,ALIAS = PS ,FORMAT = A8 ,ACTUAL = A8 ,$
```

After you create the Master File, prepare the request to produce the report. In the DEFINE command, the CHKPCK subroutine validates the salaries and moves them from the alphanumeric field, PACK_SAL, to the packed field, GOOD_PACK. The GOOD_PACK field contains either employees salaries or the error code -999. The request is:

```
DEFINE FILE TESTPACK
GOOD_PACK/P8CM = CHKPCK(8, PACK_SAL, -999, GOOD_PACK);
END

TABLE FILE TESTPACK
PRINT DEPARTMENT GOOD_PACK BY EMP_ID
END
```

The request produces the following report:

```
PAGE     1


EMP_ID     DEPARTMENT    GOOD_PACK
------     ----------    ---------
071382660  PRODUCTION      $11,000
112847612  MIS             $13,200
117593129  MIS             $18,480
119265415  PRODUCTION       $9,500
119329144  PRODUCTION      $29,700
123764317  PRODUCTION        -$999
126724188  PRODUCTION      $21,120
219984371  MIS             $18,480
326179357  MIS             $21,780
451123478  PRODUCTION        -$999
543729165  MIS              $9,000
818692173  MIS             $27,062
```

CTRAN: Translating One Character to Another

The CTRAN subroutine translates one character to another. This subroutine is especially useful for changing replacement characters to unavailable characters, or to characters that are difficult to input or unavailable on your keyboard.

Note: This subroutine is especially useful for inputting characters that are difficult to enter in PROMPT, such as “,” and “ ' “. It eliminates the need to enclose entries in single quotation marks. To use this subroutine, you need to know the decimal equivalent of the characters in internal machine representation. Printable EBCDIC characters and their decimal equivalents are listed in Appendix E, *Character Charts*.

Available on: All platforms.

Related subroutines:

HEXBYT

Syntax How to Translate One Character to Another

The syntax is

```
CTRAN(inlen, infield, decfrm, decto, output)
```

where:

inlen

Integer

Is the length in characters of the input string.

infield

Alphanumeric

Is the input string.

decfrm

Integer

Is the decimal value of the character to be translated.

decto

Integer

Is the decimal ASCII or EBCDIC value of the character to be used as a substitute for *decfrm*.

output

Alphanumeric

Is the name of the field that contains the resulting output string or the format of the output value enclosed in single quotation marks.

Example Report Request Converting Spaces to Underscores

The following example converts blank spaces in a field containing addresses to underscores:

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN3 AND COMPUTE
ALT_ADDR/A20 = CTRAN(20, ADDRESS_LN3, 32, 95, ALT_ADDR);
BY EMP_ID
WHERE TYPE EQ 'HSM'
END
```

The example produces the following report:

```
PAGE     1


EMP_ID     ADDRESS_LN3           ALT_ADDR
------     -----------           --------
117593129  RUTHERFORD NJ 07073   RUTHERFORD_NJ_07073_
119265415  NEW YORK NY 10039     NEW_YORK_NY_10039___
119329144  FREEPORT NY 11520     FREEPORT_NY_11520___
123764317  NEW YORK NY 10001     NEW_YORK_NY_10001___
126724188  FREEPORT NY 11520     FREEPORT_NY_11520___
451123478  ROSELAND NJ 07068     ROSELAND_NJ_07068___
543729165  JERSEY CITY NJ 07300  JERSEY_CITY_NJ_07300
818692173  FLUSHING NY 11354     FLUSHING_NY_11354___
```

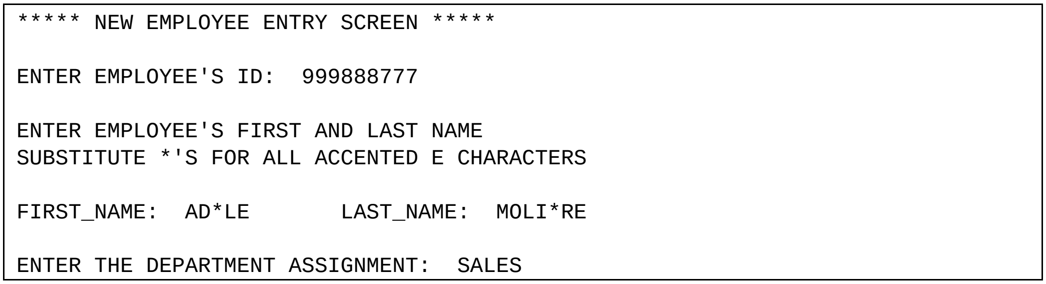
Example MODIFY Request Inserting Accented Letter E’s

This MODIFY procedure enables you to enter the names of new employees containing the accented letter È, as in the name Adèle Molière, for example. The equivalent EBCDIC code for an asterisk is 92, for an È, 159. (**Note:** If you are using the Hot Screen facility, disable it with SET SCREEN=OFF in order to display the accented letter È.)

The procedure is:

```
MODIFY FILE EMPLOYEE
CRTFORM
"***** NEW EMPLOYEE ENTRY SCREEN *****"
" "
"ENTER EMPLOYEE'S ID: <EMP_ID"
" "
"ENTER EMPLOYEE'S FIRST AND LAST NAME"
"SUBSTITUTE *'S FOR ALL ACCENTED E CHARACTERS"
" "
"FIRST_NAME: <FIRST_NAME LAST_NAME: <LAST_NAME"
" "
"ENTER THE DEPARTMENT ASSIGNMENT: <DEPARTMENT"
MATCH EMP_ID
    ON MATCH REJECT
    ON NOMATCH COMPUTE
        FIRST_NAME/A10 = CTRAN (10, FIRST_NAME, 92, 159, 'A10');
        LAST_NAME/A15 = CTRAN (15, LAST_NAME, 92, 159, 'A15');
    ON NOMATCH TYPE "FIRST_NAME: <FIRST_NAME LAST_NAME:<LAST_NAME"
    ON NOMATCH INCLUDE
DATA
END
```

A sample execution is as follows:



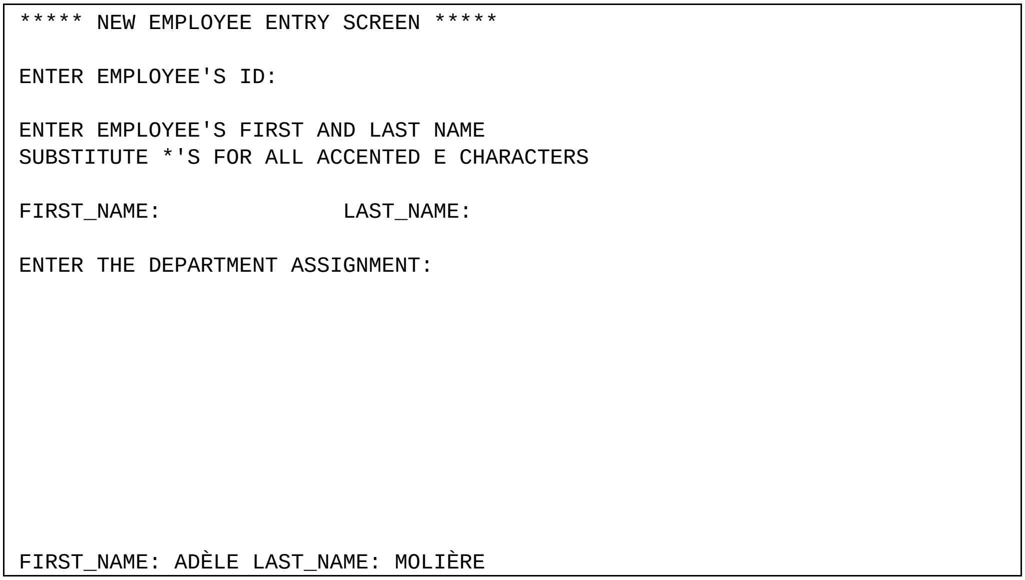
```
***** NEW EMPLOYEE ENTRY SCREEN *****

ENTER EMPLOYEE'S ID:  999888777

ENTER EMPLOYEE'S FIRST AND LAST NAME
SUBSTITUTE *'S FOR ALL ACCENTED E CHARACTERS

FIRST_NAME:  AD*LE       LAST_NAME:  MOLI*RE

ENTER THE DEPARTMENT ASSIGNMENT:  SALES
```

The procedure processes as:

1. The CRTFORM screen prompts you for an employee ID, last name, first name, and department assignment. It requests that you substitute an asterisk (*) whenever the accented letter È appears in a name.
2. You enter the following data:

```
EMPLOYEE ID:       999888777
FIRST_NAME:        AD*LE
LAST_NAME:         MOLI*RE
DEPARTMENT:        SALES
```

3. The procedure searches the data source for the employee ID. If it does not find it, it continues processing the request.
4. The CTRAN subroutine converts the asterisks into È’s in both the first and last names (ADÈLE MOLIÈRE).



```
***** NEW EMPLOYEE ENTRY SCREEN *****

ENTER EMPLOYEE'S ID:

ENTER EMPLOYEE'S FIRST AND LAST NAME
SUBSTITUTE *'S FOR ALL ACCENTED E CHARACTERS

FIRST_NAME:              LAST_NAME:

ENTER THE DEPARTMENT ASSIGNMENT:










FIRST_NAME: ADÈLE LAST_NAME: MOLIÈRE
```

5. The procedure stores the data in the data source.

Note: If you are using the Hot Screen facility, some unusual characters cannot be displayed. If Hot Screen does not support the character you chose, enter the FOCUS command

```
SET SCREEN = OFF
RETYPE
```

and redisplay the report which will appear as regular terminal output. If your terminal can display the character, the character will appear. The display of special characters depends upon your software and hardware; not all special characters may display.

Example

MODIFY Request Inserting Commas

This MODIFY procedure adds records of new employees to the EMPLOYEE data source. The PROMPT statement prompts you for data one field at a time. The CTRAN subroutine enables you to enter commas in names without having to enclose the names in single quotation marks. Instead of typing the comma, you type a semicolon, which is converted by the CTRAN subroutine into a comma. The equivalent EBCDIC code for a semicolon is 94; for a comma, 107.

The request is:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID LAST_NAME FIRST_NAME DEPARTMENT
MATCH EMP_ID
    ON MATCH REJECT
    ON NOMATCH COMPUTE
        LAST_NAME/A15 = CTRAN (15, LAST_NAME, 94, 107, 'A15');
    ON NOMATCH INCLUDE
DATA
```

A sample execution is as follows:

```
>
 EMPLOYEEFOCUS   A ON 04/19/96 AT 16.07.29
 DATA FOR TRANSACTION    1

 EMP_ID      =
224466880
 LAST_NAME   =
BRADLEY; JR.
 FIRST_NAME  =
JOHN
 DEPARTMENT  =
MIS
 DATA FOR TRANSACTION    2

 EMP_ID      =
end
 TRANSACTIONS:         TOTAL =     1  ACCEPTED=     1  REJECTED=     0
 SEGMENTS:             INPUT =     1  UPDATED =     0  DELETED =     0
>
```

The procedure processes as:

1. The procedure prompts you for an employee ID, last name, first name, and department assignment. You enter the following data:

   ```
   EMP_ID:     224466880
   LAST_NAME:  BRADLEY; JR.
   FIRST_NAME: JOHN
   DEPARTMENT: MIS
   ```

2. The procedure searches the data source for the ID 224466880. If it does not find the ID, it continues processing the transaction.

3. The CTRAN subroutine converts the semicolon in “BRADLEY; JR.” to a comma. The last name is now “BRADLEY, JR.”

4. The procedure adds the transaction to the data source.

This report displays the semicolon converted as a comma:

```
table file employee
print emp_id last_name first_name department
if emp_id is 224466880
end
```

```
 NUMBER OF RECORDS IN TABLE=        1  LINES=      1

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

  PAGE     1


  EMP_ID     LAST_NAME        FIRST_NAME  DEPARTMENT
  ------     ---------        ----------  ----------
  224466880  BRADLEY, JR.     JOHN        MIS
```

CTRFLD: Centering a Character String

The CTRFLD subroutine centers character strings within fields. The number of leading spaces is equal to or one less than the number of trailing spaces.

The CTRFLD subroutine is useful for centering the contents of a field and its report column or a heading that consists only of an embedded field. The report phrase HEADING CENTER centers each field value including trailing spaces. To center the field value without the trailing spaces, first center the value within the field using the CTRFLD subroutine.

Available on: All platforms.

Related functions and subroutines:

- LJUST
- RJUST

Note: Use of CTRFLD in a styled report (StyleSheets feature) generally negates the effect of this feature unless the item is also styled as a centered element.

Syntax How to Center a Character String

The syntax is

```
CTRFLD(infield, inlength, outfield)
```

where:

infield

Alphanumeric

Is the input field or a string enclosed in single quotation marks.

inlength

Integer

Is the length of the input and output fields. This argument must be greater than 0. (A length less than 0 can cause unpredictable results.)

outfield

Alphanumeric

Is the name of the field to which the centered output is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Centering LAST_NAME

The following example prints last names left-justified and centered:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
CENTER_NAME/A15 = CTRFLD(LAST_NAME, 15, 'A15');
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        CENTER_NAME
---------        -----------
STEVENS              STEVENS
SMITH                  SMITH
JONES                  JONES
SMITH                  SMITH
BANNING              BANNING
IRVING               IRVING
ROMANS               ROMANS
MCCOY                  MCCOY
BLACKWOOD          BLACKWOOD
MCKNIGHT           MCKNIGHT
GREENSPAN          GREENSPAN
CROSS                  CROSS
```

DA Subroutines: Converting a Date to an Integer

The DA subroutines convert dates to the corresponding number of days elapsed since December 31, 1899. By converting dates to the number of days, you can add and subtract dates and calculate the intervals between them. You can convert the results back to date format by using the DT subroutines discussed in *DT Subroutines: Converting an Integer to a Date* on page 9-87.

There are six DA subroutines; each one accepts dates in a different format.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

Available on: All platforms.

Related functions and subroutines:

- CHGDAT
- DATEDIF
- DT subroutines

Syntax How to Convert a Date to an Integer

The syntax is

`subroutine(indate, outfield)`

where:

`subroutine`

Is one of the following:

`DADMY` converts dates in day-month-year format.

`DADYM` converts dates in day-year-month format.

`DAMDY` converts dates in month-day-year format.

`DAMYD` converts dates in month-year-day format.

`DAYDM` converts dates in year-day-month format.

`DAYMD` converts dates in year-month-day format.

`indate`

Numeric

Is the input date or a field that contains the date. The date is truncated to an integer before conversion. The date format is determined by the subroutine, as explained above.

To specify the year, enter only the last two digits; the subroutine assumes the century component. If the date is invalid, the subroutine returns a 0.

outfield

Integer

Is the name of the field to which the number of days this century is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Calculating the Difference Between Two Dates

The following example shows the number of days that elapse between the time employees get raises and the time they were hired:

```
TABLE FILE EMPLOYEE
PRINT DAT_INC AS 'RAISE DATE' AND COMPUTE
DAYS_HIRED/I8 = DAYMD(DAT_INC, 'I8') - DAYMD(HIRE_DATE, 'I8');
BY LAST_NAME BY FIRST_NAME
IF DAYS_HIRED NE 0
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME  RAISE DATE  DAYS_HIRED
---------        ----------  ----------  ----------
CROSS            BARBARA       82/04/09         158
GREENSPAN        MARY          82/06/11          71
IRVING           JOAN          82/05/14         130
JONES            DIANE         82/06/01          31
MCCOY            JOHN          82/01/01         184
MCKNIGHT         ROGER         82/05/14         101
SMITH            MARY          82/01/01         184
                 RICHARD       82/05/14         130
STEVENS          ALFRED        82/01/01         578
                               81/01/01         213
```

DATEADD: Adding or Subtracting Date Units to or From a Date

The DATEADD function adds or subtracts units to or from a smart date. A unit can be any of the following:

- **Year.**
- **Month.** If you use the month unit and create invalid dates (such as February 31), DATEADD corrects them to the last day of the month. This means that adding one month to January 31 yields February 28 or February 29 (depending on whether it is a leap year), *not* February 31.
- **Day.**
- **Weekday.** If you use the weekday unit, DATEADD does not count Saturday and Sunday when adding days. This means that one weekday past a Friday is the following Monday. If your input date is a Saturday or Sunday, DATEADD adjusts it to the following Monday before performing addition or subtraction.
- **Business day.** If you use the business day unit, DATEADD uses the BUSDAYS setting and holiday file (determined by the HDAY setting) to determine which days are working days and disregards the rest. This means that if Monday is not a working day, then one business day past a Sunday is the following Tuesday. See *DATE Function Settings* on page 9-78 for more information.

DATEADD can help you:

- Compute payroll dates.
- Track and ship orders.
- Ensure correct credit card transactions.

Note: You can perform non day-based date calculations (for example YM, YQ) directly (+, -) without using these functions.

Available on: MVS, VM/CMS.

Syntax How to Add or Subtract Date Units to or From a Date

The syntax is

```
DATEADD(YYMDdate, 'unit', #units)
```

where:

YYMDdate

Date

Is any day-based new date, for example, YYMD, MDY, or JUL.

unit

Alphanumeric
Can be one of the following:

`Y` indicates year units.

`M` indicates month units.

`D` indicates day units.

`WD` indicates weekday units. This means that DATEADD disregards Saturday and Sunday when performing calculations.

`BD` indicates business day units. This means that DATEADD uses the BUSDAYS setting and the holidays file (determined from the HDAY setting) to determine which days are working days. DATEADD disregards non-working days when performing calculations.

#units

Integer

Is the number of date units you wish to add or subtract to or from the day-based new date. If this number is not a whole unit, it is rounded down to the next largest integer.

Example Adding Five Days to a Date

The following expressions add five days to a date to yield a value of 2000/01/05:

```
DATE/YYMD = '19991231';
```

```
NEWDATE/YYMD = DATEADD(DATE, 'D', 5);
```

Example Rounding With DATEADD

The number of units passed to DATEADD is always a whole unit. For example,

```
DATEADD(DATE, 'M', 1.999)
```

adds one month because the number of units is less than two.

Example Using Weekday Units

If you use weekday units and use a Saturday or Sunday as input, DATEADD adjusts the input to Monday. Thus,

```
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 one, yielding Tuesday.

Example Report Request That Determines Whether a Date Is a Business Day

The following example uses DATEADD to determine whether a date is a business day.

To run this example you need a DATE Master File and a DATE data source.

Assume the DATE Master File is as follows:

```
FILENAME = DATE, SUFFIX=FIX,$
SEGNAME=SEG1, SEGTYPE = S0,$
FIELD =  D1_YYMD, ALIAS = D1, FORMAT=YYMD,$
```

The DATE data source should have the following records:

```
19980605
19980606
```

In CMS you *must* filedef the DATE data source. For example:

```
FILEDEF DATE DISK DATE DATA A
```

In MVS, you *must* allocate the DATE data source to ddname DATE. For example:

```
DYNAM ALLOC DD DATE DA USER1.DATE.DATA SHR REU
```

The request follows:

```
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 DATA SOURCE 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 BUSINESS DAY.  THE DATA SOURCE ALSO HAS '1998/06/05,' A SATURDAY.   "
 PRINT D1_YYMD X
 IF X EQ '19980606'
END
```

The preceding request yields the following:

```
PAGE      1

 USE DATEADD TO DETERMINE WHETHER A SMARTDATE FIELD IS A BUSINESS
 DAY.  THE DATA SOURCE 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 DATA SOURCE ALSO HAS '1998/06/05,' A SATURDAY.
D1_YYMD     X
-------     -
```

DATECVT: Converting Date Formats

DATECVT converts date formats within applications without requiring intermediate calculations.

DATECVT can help you:

- Compute payroll dates.
- Track and ship orders.
- Ensure correct credit card transactions.

Available on: MVS, VM/CMS.

Related functions and subroutines:

- DA subroutines
- DT subroutines

Syntax How to Convert a Date Format

The syntax is

```
DATECVT(indate, 'infmt', 'outfmt')
```

where:

indate

Date

Is the date whose format you wish to change. If you supply an invalid old date, DATECVT returns a zero value.

infmt and *outfmt*

Alphanumeric

Can be one of the following:

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 in the *infmt* parameter function as offsets from the base date of a YYMD field (12/31/1900).

The format of the field on the left side of the equal sign must match the *outfmt* value.

Invalid formats cause DATECVT to return a zero value or blank.

Example Converting YYMD to DMY

For example,

```
field/DMY = DATECVT(indate, 'YYMD', 'DMY');
```

If the value of *indate* is 19991231 then *field* is set to the offset, which is 311299. *Indates* with old formats obey any DEFCENT and YRTHRESH values implied for that field when performing the conversion.

DATEDIF: Finding the Difference Between Two Dates

DATEDIF returns the difference between two dates in the form of a whole number, expressed in terms of units. A unit can be any of the following:

- **Year.**
- **Month.**
- **Day.**
- **Weekday.** If you use the weekday unit, DATEDIF does not count Saturday and Sunday when adding days. This means that the difference between Friday, December 21, 1999 and Monday, January 3, 2000, is one day.
- **Business day.** If you use the business day unit, DATEADD uses the BUSDAYS setting and holiday file (determined by the HDAY setting) to determine which days are working days and disregards the rest. This means that if Friday, December 31, 1999 is a holiday and Saturday and Sunday are not business days, the difference between Thursday, December 30, 1999 and Monday, January 3, 2000, is one day. See *DATE Function Settings* on page 9-78 for more information.

DATEDIF always returns a whole number. If the difference between two dates is not a whole number, say the number of years between March 2, 1996 and March 1, 1997, DATEDIF rounds down to the next largest integer. Thus the number of years between March 2, 1996 and March 1, 1997 is 0.

If you use month units, and one or both of your input dates is the end of the month, DATEDIF takes this into account. This means that the difference between January 31 and April 30 is three months, not two months.

If the to-date is before the from-date, DATEDIF returns a negative number.

DATEDIF can help you:

- Compute payroll dates.
- Track and ship orders.
- Ensure correct credit card transactions.

Note: You can perform non day-based date calculations (for example YM, YQ) directly (+, -) without using DATEDIF.

Available on: MVS, VM/CMS.

Syntax How to Return the Difference Between Two Dates

The syntax is

```
DATEDIF(fromYYMD, toYYMD, 'unit')
```

where:

fromYYMD

Date

Is the starting date from which to calculate the difference.

toYYMD

Date

Is the ending date from which to calculate the difference.

unit

Alphanumeric

Can be one of the following:

`Y` indicates year units.

`M` indicates month units.

`D` indicates day units.

`WD` indicates weekday units. This means that DATEDIF disregards Saturday and Sunday when performing calculations.

`BD` indicates business day units. This means that DATEDIF uses the BUSDAYS setting and the holidays file (determined from the HDAY setting) to determine which days are working days. DATEDIF disregards non-working days when performing calculations.

Example Rounding With DATEDIF

The following expression

```
DATEDIF(19960302, 19970301, 'Y')
```

returns 0 because the difference between March 2, 1996 and March 1, 1997 is less than a whole year.

Example Using Month Calculations

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.

The following expressions

```
DATEDIF(19990228, 19990128, 'M')
```

```
DATEDIF(19990228, 19990129, 'M')
```

```
DATEDIF(19990228, 19990130, 'M')
```

```
DATEDIF(19990228, 19990131, 'M')
```

all return a result of minus one month.

Additional examples:

`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 one year is equal to February 28th.

DATEMOV: Moving Dates to a Significant Point

DATEMOV enables you to move a date to a significant point on the calendar. DATEMOV works with smart dates only.

DATEMOV is affected by the BUSDAYS setting and the holiday file (determined by the HDAY setting). See *DATE Function Settings* on page 9-78 for more information.

DATEMOV can help you:

- Compute payroll dates.
- Track and ship orders.
- Ensure correct credit card transactions.

Available on: MVS, VM/CMS.

Syntax How to Move a Date to a Significant Point

The syntax is

```
DATEMOV(YYMDdate, 'move-point')
```

where:

YYMDdate

Date

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

Alphanumeric

Is the significant point to which you wish to move. Permissible move-points are:

`EOM` for end of month.

`BOM` for beginning of month.

`EOQ` for end of quarter.

`BOQ` for beginning of quarter.

`EOY` for end of year.

`BOY` for beginning of year.

`EOW` for end of week.

`BOW` for beginning of week.

`NWD` for next weekday.

`NBD` for next business day (affected by BUSDAYS setting and holiday files).

`PWD` for prior weekday.

`PBD` for prior business day (affected by BUSDAYS setting and holiday files).

`WD-` for a weekday or earlier.

`BD- f`or a business day or earlier (affected by BUSDAYS setting and holiday files).

`WD+` for a weekday or later.

`BD+` for a business day or later (affected by BUSDAYS setting and holiday files).

Invalid move-points result in a zero being returned.

Example Report Request Using DATEMOV

The following DEFINE statement defines a date called ADATE, which is May 7, 1998 and calculates significant points for this date:

```
DEFINE FILE CAR
ANUM/I5 WITH COUNTRY = ANUM+1;
ADATEX/YYMD 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
```

The following command sets the business days to Monday, Tuesday, Wednesday, and Thursday:

```
SET BUSDAY = _MTWT__
```

The following table request

```
TABLE FILE CAR
HEADING
"Examples of DATEMOV"
"Business days are Monday, Tuesday, Wednesday, + Thursday "
" "
"START DATE.. | 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
```

DATE Function Settings

The DATE functions (DATEADD, DATECVT, DATEDIF, and DATEMOV) enable you to manipulate smart dates. These four functions take advantage of two settings:

- The BUSDAYS setting determines which days are considered business days and which 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.
- The HDAY setting determines which file contains the list of holidays, so a single installation can support different holiday schedules.

Available on: MVS, VM/CMS.

Setting 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.

Syntax How to Set Business Days

The syntax is

```
SET BUSDAYS = smtwtfs
```

where:

smtwtfs

Is the seven-character list of days that represents your business week. The list has a position for each day from Sunday to Saturday.

- If you want a day of the week to be a business day, enter the first letter of that day in that day’s position.
- If you want a day of the week not to be a business day, enter an underscore (_) in that day’s position.

If any position within SMTWTFS is either not in its correct position or is not an underscore, FOCUS displays an error message.

Example Setting Business Days to Sunday, Tuesday, Wednesday, Friday, and Saturday

Using the example of a company that does business on Sunday, Tuesday, Wednesday, Friday, and Saturday, business days are represented as:

```
SET BUSDAYS = S_TW_FS
```

Syntax How to View the Current Setting of Business Days

The syntax is

```
? SET [ALL]
```

Setting Holidays

You can 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.

The list of holidays is defined by a file called HDAY*xxxx*.

- In MVS this file should be a member in ERRORs called HDAY*xxxx*.
- In CMS the list should be HDAY*xxxx* ERRORS.

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.

Procedure How to Define Holidays Using a Holiday File

1. Open the procedure HDAYMAKE and follow the directions to create the holiday file. (Information Builders supplies a sample Master File named HDAYDB.)

2. Execute HDAYMAKE.

3. Execute the following SET command

```
SET HDAY = xxxx
```

where:

xxxx
Is the part of the name of the holiday file after HDAY.

A sample Master File (HDAYDB) and procedure (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 procedure and execute the procedure to create the holiday file. The SET HDAY command controls the value of *xxxx* so that a single installation can support different holiday schedules.

Example Using the HDAYSTKM Holiday File

For example,

```
SET HDAY = STKM
```

reads in the holidays from member HDAYSTKM.

Syntax How to View the Current Setting of HDAY

The syntax is

```
? SET [ALL]
```

DECODE: Decoding Values

The DECODE function assigns values based on the value of an input field.

Many times the value of a field is a coded value. For example, the field SEX may have code F for female employees and M for male employees. This allows the value to be stored more efficiently (in this case, one character instead of six for female, or four for male), greatly reducing the storage requirement for the file. One method for decoding (expanding) these values is to provide a series of nested IF … THEN … ELSE phrases. For example,

```
IF SEX IS M THEN 'MALE' ELSE 'FEMALE'
```

but this can become very cumbersome and inefficient if there are numerous codes. The DECODE function facilitates the handling of codes.

There are two ways to use DECODE: you can type your values directly into the DECODE statement, or you can read your values from a separate file.

Available on: All platforms.

Syntax How to Decode Values

The syntax is

```
DECODE fieldname(code1 result1 code2 result2...[ELSE default ]);
```

where:

fieldname

Alphanumeric or Numeric

Is the name of the input field.

code

Any supported format

Is what DECODE is searching for; once it has found the specified value, it will assign the corresponding result. If the value has embedded blanks or commas, enclose it in single quotation marks.

`result`

Any supported format

Is the value to be assigned when the field has the corresponding code. If the value has embedded blanks or commas, enclose it in single quotation marks.

`default`

Any supported format

Is the value to be assigned if the code is not found among the list of codes. If this default is omitted, DECODE will assign a blank or zero for non-matching codes.

Note:

- You can use up to 40 lines to define the code and result pairs for any given DECODE expression. You can use either commas or blanks to separate the code from the result, or one pair from another.
- When explicitly coded in a procedure, you can use up to 40 lines of DECODE pairs; 39 if you also use an ELSE phrase.
- DECODE may give numeric results. Negative numbers must be enclosed in single quotation marks.
- Elements that contain either a comma or a blank must be enclosed in single quotation marks.

Syntax How to Decode Values With a File

You can also store the DECODE list in a separate file. The syntax is

```
DECODE fieldname(ddname [ELSE default ]);
```

where:

`fieldname`

Alphanumeric or Numeric

Is the name of the input field.

`ddname`

Is a logical name or a shorthand name that points to the physical file name containing the decoded values.

`default`

Any supported format

Is the value to be assigned if the code is not found among the list of codes. If this default is omitted, DECODE will assign a blank or zero for non-matching codes.

Note:

- Each record in the separate file is expected to contain one pair of elements separated by a comma or blanks.
- All data is interpreted in ASCII format on UNIX and Windows, or in EBCDIC format on MVS or CMS, and converted to the USAGE formats of the DECODE pairs.
- Leading and trailing blanks are ignored.
- The remainder of each record is also ignored and can be used for comments or other data. This convention is followed in all cases, except when the file name is HOLD. In that case the file is presumed to have been created by the FOCUS HOLD command, which writes fields in their internal format, and the DECODE pairs are interpreted accordingly. In this case, extraneous data in the record is ignored.
- If each record in the file consists of only one element, this element is interpreted as the code, and the result becomes either blanks or zero, as needed.

  This makes it possible to use the file to hold screening literals referenced in the screening condition

  ```
  IF field IS (filename)
  ```

  and as a file of literals for an IF condition specified in a computational expression. For example:

  ```
  TAKE = DECODE SELECT (filename ELSE 1);
  VALUE = IF TAKE IS 0 THEN... ELSE...;
  ```

  TAKE will be 0 for SELECT values found in the literal file and 1 in all other cases. The VALUE computation is carried out as if the expression had been:

  ```
  IF SELECT (filename) THEN... ELSE...;
  ```

- When using DECODE via a file, you can have up to 32,767 characters in the file.

Example Report Request Assigning Job Categories Based on CURR_JOBCODE

The following examples use EDIT to extract the first character of the field CURR_JOBCODE. It then uses DECODE to create a value for the field JOB_CATEGORY.

```
TABLE FILE EMPLOYEE
PRINT CURR_JOBCODE AND COMPUTE
DEPX_CODE/A1 = EDIT(CURR_JOBCODE,'9$$') ; NOPRINT AND COMPUTE
JOB_CATEGORY/A15 = DECODE DEPX_CODE(A 'ADMINISTRATIVE'
                                    B 'DATA PROCESSING') ;
BY LAST_NAME
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        CURR_JOBCODE  JOB_CATEGORY
---------        ------------  ------------
BANNING          A17           ADMINISTRATIVE
BLACKWOOD        B04           DATA PROCESSING
CROSS            A17           ADMINISTRATIVE
GREENSPAN        A07           ADMINISTRATIVE
IRVING           A15           ADMINISTRATIVE
JONES            B03           DATA PROCESSING
MCCOY            B02           DATA PROCESSING
MCKNIGHT         B02           DATA PROCESSING
ROMANS           B04           DATA PROCESSING
SMITH            B14           DATA PROCESSING
                 A01           ADMINISTRATIVE
STEVENS          A07           ADMINISTRATIVE
```

Example Report Request Reading DECODE Values From a File

The following report request has two parts. The first part creates a file with a list of the employee IDs for the employees who have taken classes. The second part reads this file and assigns 0 to those employees who have taken classes and 1 to those employees who have not. (Notice that the HOLD file contains only one column of values; therefore DECODE assigns the value 0 to an employee when their EMP_ID appears in the file and 1 when EMP_ID does not appear in the file.)

```
TABLE FILE EDUCFILE
PRINT EMP_ID
ON TABLE HOLD
END
```

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND LAST_NAME AND FIRST_NAME AND
COMPUTE NOT_IN_LIST/I1 = DECODE EMP_ID(HOLD ELSE 1);
END
```

This request produces the following report:

```
PAGE     1


EMP_ID     LAST_NAME        FIRST_NAME  NOT_IN_LIST
------     ---------        ----------  -----------
071382660  STEVENS          ALFRED                0
112847612  SMITH            MARY                  0
117593129  JONES            DIANE                 0
119265415  SMITH            RICHARD               0
119329144  BANNING          JOHN                  1
123764317  IRVING           JOAN                  1
126724188  ROMANS           ANTHONY               1
219984371  MCCOY            JOHN                  1
326179357  BLACKWOOD        ROSEMARIE             0
451123478  MCKNIGHT         ROGER                 0
543729165  GREENSPAN        MARY                  1
818692173  CROSS            BARBARA               0
```

DMY, MDY, YMD: Calculating the Difference Between Two Dates

The DMY, MDY, and YMD functions calculate the difference between two dates in integer, alphanumeric, or packed format:

Available on: All platforms.

Related functions and subroutines:

YM

Syntax How to Calculate the Difference Between Two Dates

The syntax is

`function(begin, end)`

where:

`function`

Is one of the following:

`DMY` calculates the difference between two dates in day-month-year order.

`MDY` calculates the difference between two dates in month-day-year order.

`YMD` calculates the difference between two dates in year-month-day order.

begin

Numeric

Is the beginning date. You may supply the actual date or the name of a field that contains the date.

end

Numeric

Is the end date. You may supply the actual date or the name of a field that contains the date.

Example Report Request Calculating Number of Days Between Start Date and First Pay Raise

The following example calculates the number of days between employees’ start dates and their first pay raise:

```
TABLE FILE EMPLOYEE
SUM HIRE_DATE FST.DAT_INC AS 'FIRST PAY,INCREASE' AND COMPUTE
DIFF/I4 = YMD(HIRE_DATE, FST.DAT_INC) ; AS 'DAYS,BETWEEN'
BY EMP_ID
END
```

The example produces the following report:

```
PAGE     1


                      FIRST PAY  DAYS
EMP_ID     HIRE_DATE  INCREASE   BETWEEN
------     ---------  ---------  -------
071382660   80/06/02   82/01/01      578
112847612   81/07/01   82/01/01      184
117593129   82/05/01   82/06/01       31
119265415   82/01/04   82/05/14      130
119329144   82/08/01   82/08/01        0
123764317   82/01/04   82/05/14      130
126724188   82/07/01   82/07/01        0
219984371   81/07/01   82/01/01      184
326179357   82/04/01   82/04/01        0
451123478   82/02/02   82/05/14      101
543729165   82/04/01   82/06/11       71
818692173   81/11/02   82/04/09      158
```

DOWK and DOWKL: Finding the Day of the Week

The DOWK and DOWKL subroutines find the day of the week (Sunday through Saturday) of dates. The DOWK subroutine returns the day as a 3-letter abbreviation; to display the full name of the day, specify DOWKL instead.

Available on: All platforms.

Syntax How to Find the Day of the Week

The syntax is

```
DOWK(indate, outfield)
```

or

```
DOWKL(indate, outfield)
```

where:

indate

Numeric

Is the input date in year-month-day format. If the date is not valid, the subroutine returns spaces. If the date specifies a 2-digit year and DEFCENT and YRTHRESH values have not been set, the subroutine assumes the 20th century.

outfield

DOWK: A4

DOWKL: A12

Is the name of the field to which the day of the week is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Finding the Day of the Week

The following example shows on which day of the week employees were hired:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND HIRE_DATE AND COMPUTE
DATED/A4 = DOWK(HIRE_DATE, DATED);
END
```

The example produces the following report:

```
PAGE     1


EMP_ID     HIRE_DATE  DATED
------     ---------  -----
071382660   80/06/02  MON
112847612   81/07/01  WED
117593129   82/05/01  SAT
119265415   82/01/04  MON
119329144   82/08/01  SUN
123764317   82/01/04  MON
126724188   82/07/01  THU
219984371   81/07/01  WED
326179357   82/04/01  THU
451123478   82/02/02  TUE
543729165   82/04/01  THU
818692173   81/11/02  MON
```

DT Subroutines: Converting an Integer to a Date

The DT subroutines convert numbers representing the days elapsed since December 31, 1899 to corresponding dates. The DT subroutines are useful when you are performing arithmetic on dates converted to the number of days (see *DA Subroutines: Converting a Date to an Integer* on page 9-66). The DT subroutines convert the result back to date format.

There are six DT subroutines; each one converts the numbers into dates of a different format.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

Available on: All platforms.

Related functions and subroutines:

- CHGDAT
- DA subroutines
- DATEDIF

Syntax How to Convert Integers to Dates

The syntax is

subroutine(number, *outfield*)

where:

subroutine

Is one of the following:

DTDMY converts numbers to day-month-year dates.

DTDYM converts numbers to day-year-month dates.

DTMDY converts numbers to month-day-year dates.

DTMYD converts numbers to month-year-day dates.

DTYDM converts numbers to year-day-month dates.

DTYMD converts numbers to year-month-day dates.

number

Numeric

Is the number of days. The number is truncated to an integer.

outfield

Integer

Is the name of the field to which the corresponding date is returned. The date format is determined by the subroutine, as explained above. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Converting Integer to Date

The following example takes a date that has been converted to the number of days (34650) and converts it back to the corresponding date, in month-day-year format:

```
-* THIS PROCEDURE CONVERTS HIRE_DATE, WHICH IS IN I6YMD FORMAT,
-* TO A DATE IN I8MDYY FORMAT.
-* FIRST IT USES THE DAYMD SUBROUTINE TO CONVERT HIRE_DATE
-* TO A NUMBER OF DAYS.
-* THEN IT USES THE DTMDY SUBROUTINE TO CONVERT THIS NUMBER OF
-* DAYS TO I8MDYY FORMAT
-*
DEFINE FILE EMPLOYEE
NEWF/I8 WITH EMP_ID=DAYMD(HIRE_DATE,NEWF);
NEW_HIRE_DATE/I8MDYY WITH EMP_ID=DTMDY(NEWF,NEW_HIRE_DATE);
END
TABLE FILE EMPLOYEE
PRINT HIRE_DATE NEW_HIRE_DATE
BY FN BY LN
END
```

The example produces the following report:

```
PAGE     1

FIRST_NAME  LAST_NAME        HIRE_DATE  NEW_HIRE_DATE
----------  ---------        ---------  -------------
ALFRED      STEVENS           80/06/02     06/02/1980
ANTHONY     ROMANS            82/07/01     07/01/1982
BARBARA     CROSS             81/11/02     11/02/1981
DIANE       JONES             82/05/01     05/01/1982
JOAN        IRVING            82/01/04     01/04/1982
JOHN        BANNING           82/08/01     08/01/1982
            MCCOY             81/07/01     07/01/1981
MARY        GREENSPAN         82/04/01     04/01/1982
            SMITH             81/07/01     07/01/1981
RICHARD     SMITH             82/01/04     01/04/1982
ROGER       MCKNIGHT          82/02/02     02/02/1982
ROSEMARIE   BLACKWOOD         82/04/01     04/01/1982
```

EDIT: Converting the Format of a Field

You can use the EDIT function to convert an alphanumeric field that contains numeric characters to numeric format, or to convert a numeric field to alphanumeric format. This is useful when you need to manipulate the field using a command or keyword that requires a particular format.

Note: The EDIT function also extracts characters from or adds characters to an alphanumeric string.

Available on: All platforms.

Related functions and subroutines:

FTOA

Syntax How to Convert Field Formats

The syntax is

```
EDIT(fieldname);
```

where:

fieldname

Alphanumeric or Numeric

Is the field name, enclosed in parentheses.

When you use EDIT to assign the converted value to a field, the format of the new field must correspond to the format of the returned value. For example, if you use EDIT to convert a numeric field to alphanumeric format, and then assign the resulting value to an alphanumeric field, you must give the new field an alphanumeric format as follows:

```
DEFINE ALPHAPRICE/A6 = EDIT(PRICE);
```

When converting an alphanumeric field to numeric format, a sign or decimal point in the field is accepted and is reflected in the value stored in the numeric field.

When converting a floating-point or packed-decimal field to alphanumeric format, EDIT removes the sign, the decimal point, and any number to the right of the decimal point. It then right-justifies the remaining digits and adds leading zeros to the specified field length. Also, converting a number with more than nine significant digits in floating-point or packed-decimal format may produce an incorrect result.

Example Report Request That Converts HIRE_DATE to Alphanumeric Format

The following example uses the CHGDAT subroutine to spell out an employee’s hire date. However, CHGDAT expects its input date to be in alphanumeric format, and the HIRE_DATE field is numeric. Therefore, this report request defines a hidden field, ALPHA_HIRE, containing the contents of HIRE_DATE converted to alphanumeric format. Then CHGDAT uses ALPHA_HIRE as input.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
ALPHA_HIRE/A17 = EDIT(HIRE_DATE); NOPRINT AND COMPUTE
HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX',ALPHA_HIRE,'A17');
BY LAST_NAME BY FIRST_NAME
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME  HIRE_DATE  HIRE_MDY
---------        ----------  ---------  --------
BANNING          JOHN         82/08/01  AUGUST 01 1982
BLACKWOOD        ROSEMARIE    82/04/01  APRIL 01 1982
CROSS            BARBARA      81/11/02  NOVEMBER 02 1981
GREENSPAN        MARY         82/04/01  APRIL 01 1982
IRVING           JOAN         82/01/04  JANUARY 04 1982
JONES            DIANE        82/05/01  MAY 01 1982
MCCOY            JOHN         81/07/01  JULY 01 1981
MCKNIGHT         ROGER        82/02/02  FEBRUARY 02 1982
ROMANS           ANTHONY      82/07/01  JULY 01 1982
SMITH            MARY         81/07/01  JULY 01 1981
                 RICHARD      82/01/04  JANUARY 04 1982
STEVENS          ALFRED       80/06/02  JUNE 02 1980
```

EDIT: Extracting or Adding Characters

You can use the EDIT function to extract characters from or add characters to an alphanumeric string.

Note: The EDIT function also converts the format of a field.

Available on: All platforms.

Related functions and subroutines:

SUBSTR

Syntax How to Extract or Add Characters

The syntax is

`EDIT(fieldname, 'mask');`

where:

fieldname

Alphanumeric or Numeric

Is the name of the source field.

mask

Alphanumeric

Is a string, enclosed in single quotation marks (').

EDIT compares the characters in the mask to the characters in the source field. When it encounters a 9 in the mask, EDIT copies the corresponding character from the source field to the new field. When it encounters a $ (dollar sign) in the mask, EDIT ignores the corresponding character in the source field. When it encounters any other character in the mask, EDIT copies that character to the corresponding position in the new field.

Example Report Request Extracting First Initial of FIRST_NAME and Adding Dashes to EMP_ID

The following request shows how you can use masking to extract the first initial from FIRST_NAME and add dashes to EMP_ID. EMP_ID has the format A9; FIRST_NAME has the format A10. The request produces two new fields, FIRST_INIT and EMPIDEDIT, which contain the first initial, and an employee ID with dashes added to enhance readability, respectively.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
FIRST_INIT/A1 = EDIT(FIRST_NAME, '9$$$$$$$$$');
EMPIDEDIT/A11 = EDIT(EMP_ID, '999-99-9999');
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following report.

```
PAGE     1


LAST_NAME        FIRST_INIT  EMPIDEDIT
---------        ----------  ---------
SMITH            M           112-84-7612
JONES            D           117-59-3129
MCCOY            J           219-98-4371
BLACKWOOD        R           326-17-9357
GREENSPAN        M           543-72-9165
CROSS            B           818-69-2173
```

EXP: Raising “e” to the Nth Power

The EXP subroutine raises the value “e” (approximately 2.72) to any power you specify. This subroutine is the inverse of the LOG function, which returns an argument’s logarithm.

The subroutine calculates the answer by adding terms of an infinite series. If a term adds less than .000001 percent to the sum, the subroutine ends the calculation and returns the result as a double-precision number.

Available on: MVS, VM/CMS, UNIX, OpenVMS, WebFOCUS.

Related subroutines:

LOG

Syntax How to Raise “e” to the Nth Power

The syntax is

`EXP(`*`power, outfield`*`)`

where:

`power`

Numeric

Is the power that “e” is being raised to.

`outfield`

Double-precision

Is the name of the field that contains the result. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Raising “e” to the Nth Power

The following Dialogue Manager procedure raises “e” to the power you specify and returns the result rounded to the nearest integer (the 0.5 added to the result is a rounding constant). To determine “e” to the third power, set the &POW variable to 3.

```
-SET &POW = '3';
-SET &RESULT = EXP(&POW, 'D15.3') + 0.5;
-TYPE E TO THE &POW POWER IS APPROXIMATELY &RESULT
```

The result is 20:

```
E TO THE 3 POWER IS APPROXIMATELY 20
```

EXPN: Evaluating Scientific Notation

The EXPN function evaluates an argument expressed in scientific notation.

Available on: MVS, VM/CMS, UNIX, OpenVMS, WebFOCUS.

Syntax

How to Evaluate Scientific Notation

The syntax is

```
EXPN(argument)
```

where:

argument

Is the value on which the function operates and should have the following format

```
n.nn {E|D} {+|-} p
```

where:

n.nn is a numeric constant that consists of a whole number component, followed by a decimal point, followed by a fractional component.

{E|D} denotes scientific notation. E and D are interchangeable.

p is the power of 10 to which you want to raise the number.

You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. The expression may call a function or a subroutine.

For example, you can use scientific notation to express 103 as:

```
1.03E+2
```

FEXERR: Retrieving FOCUS Error Messages

The FEXERR subroutine retrieves a specified FOCUS error message. FOCUS error messages may consist of up to four lines of text:

- The first line contains the message.
- The remaining three may contain a detailed explanation, if it exists.

The subroutine retrieves the first line, the message portion. This subroutine is especially useful in procedures when the display of output messages is suppressed for a FOCUS session. Examples of commands that suppress messages are the CMS halt typing command (SET CMSTYPE HT) or the FOCUS terminal output command (SET TRMOUT=OFF).

Available on: MVS, VM/CMS.

Syntax How to Retrieve FOCUS Error Messages

The syntax is

```
FEXERR(nnnnn, 'A72')
```

where:

nnnnn

Numeric

Is the FOCUS error number, up to five digits.

'A72'

Is the format of the output value, enclosed in single quotation marks, which contains the retrieved message. The maximum length of FOCUS error messages is 72 characters. For Maintain, specify the field name.

Example Retrieving FOCUS Error Messages

This Dialogue Manager procedure initiates a global variable (&&MSGVAR) to store the retrieved message as a concatenated string, assigns the FOCUS error number 650 to a local variable (&ERR), and displays the message. The FEXERR subroutine retrieves the message for the error number specified as a variable.

The procedure is:

```
-SET &ERR = 650;
-SET &&MSGVAR = FEXERR(&ERR, 'A72');
-TYPE &&MSGVAR
```

When you execute this procedure, it displays the message for FOCUS error number 650:

```
(FOC650) THE DISK IS NOT ACCESSED
>
```

FINDMEM: Finding a Member of a Partitioned Data Set

The FINDMEM subroutine, used on MVS or batch only, determines if a specific member of a partitioned data set (PDS) exists. This subroutine is especially useful in Dialogue Manager procedures.

In order to use this subroutine, the PDS must be allocated to a ddname, because the ddname is specified in the subroutine call. You can search multiple partitioned data sets with one subroutine call if the partitioned data sets are concatenated to one ddname.

Available on: MVS.

Related functions and subroutines:

GETPDS

Syntax How to Find a Member of a Partitioned Data Set

The syntax is

```
FINDMEM(ddname, member, outfield)
```

where:

ddname

A8

Is the ddname to which the PDS is allocated. This argument must be eight characters long or a variable. If you are using a literal for this argument, enclose it in single quotation marks. If it is less than eight characters, pad the literal with trailing blanks.

member

A8

Is the member you are searching for. This argument must be eight characters long. If you are using a literal for this argument that has less than eight characters, pad the literal with trailing blanks.

outfield

A1

Is the name of the field that contains the result of the search: Y, N, or E. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

The subroutine searches the PDS for a specified member and returns the letter Y, N, or E:

`Y`

The member exists in the PDS.

`N`

The member does not exist in the PDS.

`E`

An error occurred. This can occur for two reasons:

1. The data set is not allocated to the ddname.
2. The data set allocated to the ddname is not a PDS (and may be a sequential file).

Example Finding the Member of a Partitioned Data Set

This Dialogue Manager procedure executes a report request if the EMPLOYEE Master File exists. The FINDMEM subroutine searches the PDS allocated to ddname MASTER for the EMPLOYEE Master File. If the subroutine does not find the description, the procedure returns the appropriate message.

The procedure is:

```
-SET &FINDCODE = FINDMEM('MASTER  ', 'EMPLOYEE', 'A1');
-IF &FINDCODE EQ 'N' GOTO NOMEM;
-IF &FINDCODE EQ 'E' GOTO NOPDS;
-TYPE MEMBER EXISTS, RETURN CODE = &FINDCODE
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY LAST_NAME BY FIRST_NAME
WHERE RECORDLIMIT EQ 4
END
-EXIT
-NOMEM
-TYPE EMPLOYEE NOT FOUND IN MASTER FILE PDS
-EXIT
-NOPDS
-TYPE ERROR OCCURRED IN SEARCH
-TYPE CHECK IF FILE IS A PDS ALLOCATED TO DDNAME MASTER
-EXIT
```

In this sample execution, the procedure finds the member and displays the report:

```
MEMBER EXISTS, RETURN CODE = Y
>   NUMBER OF RECORDS IN TABLE=        4  LINES=      4

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

PAGE     1


LAST_NAME        FIRST_NAME         CURR_SAL
---------        ----------         --------
JONES            DIANE            $18,480.00
SMITH            MARY             $13,200.00
                 RICHARD           $9,500.00
STEVENS          ALFRED           $11,000.00
```

FTOA: Converting a Number to Alphanumeric Format

The FTOA subroutine converts numbers from numeric format to alphanumeric format.

The EDIT function also converts numbers from numeric to alphanumeric format, but there are differences between FTOA and EDIT:

- FTOA retains the decimal portions of numbers, whereas EDIT truncates numbers to integers.
- FTOA stores numbers right-justified with leading spaces, whereas EDIT stores numbers right-justified with leading zeros.
- FTOA enables you to add edit options to the converted number; whereas EDIT does not.
- FTOA can process any number up to 16 digits; EDIT can process any number up to nine digits and certain numbers up to ten digits. (The limit for EDIT is due to the internal representation of the number as a 4-byte integer.)

Available on: All platforms.

Syntax How to Convert Numbers to Characters

The syntax is

```
FTOA(number, '(format)', outfield)
```

where:

number

Numeric

Is the number to be converted or the field containing the number.

'(*format*) '

Alphanumeric

Is the format of the number as it is stored in numeric format, enclosed in both single quotation marks and parentheses. Only F and D formats are supported. Include any edit options that you want to appear in the output.

Note: If you are using a field for this argument, specify the field name without quotation marks or parentheses. The values in the field must be enclosed in parentheses.

outfield

Alphanumeric

Is the name of the field to which the number in alphanumeric format is returned. This argument can also be the format of the output value, enclosed in single quotation marks. The length of this argument must be greater than the length of the *number* argument and must account for edit options and a possible negative sign. The D format automatically supplies commas.

Example Report Request Converting GROSS Salary to Alphanumeric Format

The following example converts the GROSS salary field from decimal to alphanumeric format:

```
TABLE FILE EMPLOYEE
PRINT GROSS AND COMPUTE
ALPHA_GROSS/A14 = FTOA(GROSS, '(D12.2)', ALPHA_GROSS);
BY LAST_NAME
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME                  GROSS  ALPHA_GROSS
---------                  -----  -----------
BLACKWOOD              $1,815.00        1,815.00
CROSS                  $2,255.00        2,255.00
IRVING                 $2,238.50        2,238.50
JONES                  $1,540.00        1,540.00
MCKNIGHT               $1,342.00        1,342.00
ROMANS                 $1,760.00        1,760.00
SMITH                  $1,100.00        1,100.00
STEVENS                  $916.67          916.67
```

GETPDS: Determining if a Member of a Partitioned Data Set Exists

The GETPDS subroutine determines if a specific member of a partitioned data set (PDS) exists and, if so, returns the PDS name. This subroutine is especially useful in Dialogue Manager procedures. The FINDMEM subroutine is almost identical to the GETPDS subroutine, except that the GETPDS subroutine provides either the PDS name or different status codes.

In order to use this subroutine, the PDS must be allocated to a ddname, because the ddname is specified in the subroutine call. You can search multiple partitioned data sets with one subroutine call if the partitioned data sets are concatenated to one ddname.

Available on: MVS.

Related functions and subroutines:

FINDMEM

Syntax How to Determine if a Member Exists

The syntax is

```
GETPDS(ddname, member, outfield)
```

where:

ddname

A8

Is the ddname to which the PDS is allocated. This argument must be eight characters long or a variable. If you are using a literal for this argument, enclose it in single quotation marks. If it is less than eight characters, pad the literal with trailing blanks.

member

A8

Is the member you are searching for. This argument must be eight characters long. If you are using a literal for this argument that has less than eight characters, pad the literal with trailing blanks.

outfield

A44

Is the name of the field that contains the result of the search. This argument must be 44 characters long, because the maximum length for a PDS name is 44. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

The subroutine searches the PDS for a specified member and returns one of four values:

PDS name

If the specified member exists, the PDS name that contains it.

`*D`

The ddname is not assigned (allocated) to a data set.

`*M`

The member does not exist in the PDS.

`*E`

An error occurred. This often occurs because the data set allocated to the ddname is not a PDS (and may be a sequential file).

Example Determining if a Member Exists

This Dialogue Manager procedure returns the name of the PDS if the EMPLOYEE Master File exists. The GETPDS subroutine searches the PDS allocated to ddname MASTER for the EMPLOYEE Master File.

```
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = 'EMPLOYEE';
-SET &PNAME = '                                            ';
-SET &PNAME = GETPDS(&DDNAME,&MEMBER,&PNAME);
-IF &PNAME EQ '*D' THEN GOTO DDNOAL;
-IF &PNAME EQ '*M' THEN GOTO MEMNOF;
-IF &PNAME EQ '*E' THEN GOTO DDERROR;
-*
-TYPE MEMBER &MEMBER IS FOUND IN
-TYPE THE PDS &PNAME
-TYPE ALLOCATED TO &DDNAME
-*
-EXIT
-DDNOAL
-*
-TYPE DDNAME &DDNAME NOT ALLOCATED
-*
-EXIT
-MEMNOF
-*
-TYPE MEMBER &MEMBER NOT FOUND UNDER DDNAME &DDNAME
-*
-EXIT
-DDERROR
-*
-TYPE ERROR IN GETPDS; DATA SET PROBABLY NOT A PDS.
-*
-EXIT
```

A sample execution is:

```
MEMBER EMPLOYEE IS FOUND IN
THE PDS USER1.MASTER.DATA
ALLOCATED TO MASTER
>  >
```

Example Using GETPDS With TED

In this example, the GETPDS subroutine searches for a specified member in the production MASTER.DATA partitioned data set and returns the PDS name. The DYNAM commands copy the specified member from the production PDS to your local PDS. Then, the TED editor enables you to edit the member. The ddnames are allocated earlier in the session: the production PDS is allocated to the ddname MASTER; your local PDS to ddname MYMASTER.

```
-* If the MASTER file in question is in the 'production' pds, it must
-* be copied to a 'local' pds, which has been allocated previously to the
-* ddname MYMASTER before any changes can be made.
-* Assume the MASTER in question is supplied via a -CRTFORM, with
-* a length of 8 characters, as &MEMBER
-*
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = &MEMBER;
-SET &PNAME = '                                            ';
-SET &PNAME = GETPDS(&DDNAME,&MEMBER,&PNAME);
-IF &PNAME EQ '*D' OR '*M' OR '*E' THEN GOTO DDERROR;
-*
DYNAM ALLOC FILE XXXX DA -
   &PNAME MEMBER &MEMBER SHR
DYNAM COPY XXXX MYMASTER MEMBER &MEMBER
-RUN
TED MYMASTER(&MEMBER)
-EXIT
-*
-DDERROR
-*
-TYPE Error in GETPDS; Check allocation for &DDNAME for
-TYPE proper allocation.
-*
-EXIT
```

Earlier in the FOCUS session, allocate the ddnames:

```
>  > tso alloc f(master) da('wibfoc.p7009505.master.data') shr
>  > tso alloc f(mymaster) da('user1.master.data') shr
```

After you execute the procedure, specify the EMPLOYEE member. The member is copied to your local PDS and you enter TED.

```
PLEASE SUPPLY VALUES REQUESTED

MEMBER=  > employee


MYMASTER(EMPLOYEE)                    SIZE=37    LINE=0

00000 * * * TOP OF FILE * * *
00001 FILENAME=EMPLOYEE, SUFFIX=FOC
00002 SEGNAME=EMPINFO,  SEGTYPE=S1
00003  FIELDNAME=EMP_ID,       ALIAS=EID,     FORMAT=A9,       $
00004  FIELDNAME=LAST_NAME,    ALIAS=LN,      FORMAT=A15,      $
00005  FIELDNAME=FIRST_NAME,   ALIAS=FN,      FORMAT=A10,      $
00006  FIELDNAME=HIRE_DATE,    ALIAS=HDT,     FORMAT=I6YMD,    $
00007  FIELDNAME=DEPARTMENT,   ALIAS=DPT,     FORMAT=A10,      $
```

Example Using GETPDS With Query Commands

Suppose you wanted to review the attributes of the PDS that contained a specific member. This Dialogue Manager procedure searches for the EMPLOYEE member in the PDS allocated to the ddname MASTER and, based on its existence, allocates the PDS name to the ddname TEMPMAST. Dialogue Manager MVS system variables are used to display the attributes.

```
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = 'EMPLOYEE';
-SET &PNAME = '                                            ';
-SET &PNAME = GETPDS(&DDNAME,&MEMBER,&PNAME);
-IF &PNAME EQ '*D' OR '*M' OR '*E' THEN GOTO DDERROR;
-*
DYNAM ALLOC FILE TEMPMAST DA -
   &PNAME SHR
-RUN
-? MVS DDNAME TEMPMAST
-TYPE The data set attributes include:
-TYPE Data set name is: &DSNAME
-TYPE Volume is: &VOLSER
-TYPE Disposition is: &DISP
-EXIT
-*
-DDERROR
-TYPE Error in GETPDS; Check allocation for &DDNAME for
-TYPE proper allocation.
-*
-EXIT
```

A sample execution follows:

```
>  THE DATA SET ATTRIBUTES INCLUDE:
DATA SET NAME IS: USER1.MASTER.DATA
VOLUME IS: USERMO
DISPOSITION IS: SHR
>
```

When you execute this procedure, it searches the PDS allocated to ddname MASTER for the member EMPLOYEE. Since the procedure locates the member, it displays the attributes for the MASTER PDS.

GETTOK: Getting a Token From a String

The GETTOK subroutine divides a character string where a specific character, called the delimiter, occurs in the string. It then returns one of the substrings, called a token.

For example, suppose you want to extract the fourth word from a sentence. The subroutine divides the sentence into words using spaces as delimiters, then extracts the fourth word. If the string is not divided by a delimiter character, use the PARAG subroutine.

Available on: All platforms.

Related functions and subroutines:

PARAG

Syntax How to Divide a Character String

The syntax is

```
GETTOK(infield, inlen, toknum, 'delim', outlen, outfield)
```

where:

infield

Alphanumeric

Is the field containing the parent character string.

inlen

Integer

Is the length of the parent string. If this argument is less than or equal to 0, the subroutine returns spaces.

toknum

Integer

Is the number of the token you want extracted. If this argument is positive, the tokens are numbered from left to right. If this argument is negative, the tokens are numbered from the right to left (for example, an argument of -2 indicates the second to the last token in the string). If this argument is 0, the subroutine returns spaces. Leading and trailing null tokens are ignored.

delim

Alphanumeric

Is the delimiter character in the parent string, enclosed in single quotation marks. If you specify more than one character, only the first character is used.

outlen

Integer

Is the maximum size of the token. If this argument is less than or equal to 0, the subroutine returns spaces. If the token is longer than this argument, it is truncated; if it is shorter, it is padded with trailing spaces.

outfield

Alphanumeric

Is the name of the field to which the token is returned. This argument can also be the format of the output value, enclosed in single quotation marks. **Note:** The delimiter is not included in the token.

Tip: In Dialogue Manager, to prevent the conversion of a delimiter blank character (`'  '`) to a double precision zero, include a non-numeric character after the blank (for example, `' %'`). GETTOK uses only the first character (the blank) as a delimiter and the extra character (%) prevents conversion to double precision.

Example Report Request Extracting Zip Code From Address

The following example uses a single blank space as a delimiter to break an address line into tokens and returns the last token, which is the zip code:

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN3 AND COMPUTE
LAST_TOKEN/A10 = GETTOK(ADDRESS_LN3, 20, -1, ' ', 10, LAST_TOKEN) ;
AS 'LAST TOKEN,(ZIP CODE)'
WHERE TYPE EQ 'HSM'
END
```

The example produces the following report:

```
PAGE     1

                      LAST TOKEN
ADDRESS_LN3           (ZIP CODE)
-----------           ----------
RUTHERFORD NJ 07073   07073
NEW YORK NY 10039     10039
FREEPORT NY 11520     11520
NEW YORK NY 10001     10001
FREEPORT NY 11520     11520
ROSELAND NJ 07068     07068
JERSEY CITY NJ 07300  07300
FLUSHING NY 11354     11354
```

GETUSER: Retrieving the User ID

The GETUSER subroutine retrieves the user ID (userid) of the connected user.

In MVS FOCUS, it can also retrieve the name of an MVS batch job if you run it from the batch job. To retrieve a logon ID for MSO, use the MSOINFO subroutine described in the *FOCUS for IBM Mainframe Multi-Session Option Installation and Technical Reference Guide*.

Available on: MVS, VM/CMS, UNIX, OpenVMS, WebFOCUS.

Syntax How to Retrieve the User ID

The syntax is

```
GETUSER(outfield)
```

where:

outfield

Alphanumeric eight bytes

Is the name of the field that contains the user ID. Specify a field that is eight bytes long. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Returning User ID of Person Executing Request

The following request returns employees’ department assignments and salaries; the report heading returns the user ID of the person executing the request:

```
DEFINE FILE EMPLOYEE
USERID/A8 WITH EMP_ID = GETUSER(USERID);
END
TABLE FILE EMPLOYEE
HEADING
    "SALARY REPORT RUN FROM USERID <USERID"
    " "
PRINT DEPARTMENT CURR_SAL
BY LAST_NAME BY FIRST_NAME
END
```

The request produces the following report:

```
PAGE     1

SALARY REPORT RUN FROM USERID USER1

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
```

GREGDT: Converting From Julian to Gregorian Format

The GREGDT subroutine converts dates in Julian format to year-month-day format. Dates in Julian format are 5- or 7-digit numbers. The first two or four digits are the year; the last three digits are the number of the day counting from January 1. For example, January 1, 1987 in Julian format is 87001, and December 31, 1987 is 87365. For century display, dates in Gregorian format are 8-digit numbers.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

Available on: All platforms.

Related functions and subroutines:

JULDAT

Syntax How to Convert Julian Format Dates to Gregorian Format

The syntax is

```
GREGDT(indate, outfield)
```

where:

indate

Numeric

Is the Julian date, which is truncated to an integer before conversion. Each value must be a 5- or 7-digit number after truncation. The first two or four digits represent the year (YMD or YYMD), the last three digits must be between 001 and 365 (366 for a leap year). If the date is invalid, the subroutine returns a 0.

outfield

Integer

Is the name of the field to which the date in year-month-day format is returned. This argument can also be the format of the output value, enclosed in single quotation marks (I6 or I8). For Maintain, specify the field name.

Example Report Request Converting Date to Julian and Gregorian Date

The following example converts HIRE_DATE to both a Julian date and a Gregorian date:

```
TABLE FILE EMPLOYEE
ON TABLE SET SQUEEZE ON
PRINT DEPARTMENT AND HIRE_DATE AND
COMPUTE JULIAN/I6 = JULDAT(HIRE_DATE, JULIAN); AND
COMPUTE GREG_DATE/I6 = GREGDT(JULIAN, 'I6'); BY LAST_NAME
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        DEPARTMENT  HIRE_DATE  JULIAN  GREG_DATE
---------        ----------  ---------  ------  ---------
BANNING          PRODUCTION   82/08/01   82213     820801
BLACKWOOD        MIS          82/04/01   82091     820401
CROSS            MIS          81/11/02   81306     811102
GREENSPAN        MIS          82/04/01   82091     820401
IRVING           PRODUCTION   82/01/04   82004     820104
JONES            MIS          82/05/01   82121     820501
MCCOY            MIS          81/07/01   81182     810701
MCKNIGHT         PRODUCTION   82/02/02   82033     820202
ROMANS           PRODUCTION   82/07/01   82182     820701
SMITH            MIS          81/07/01   81182     810701
                 PRODUCTION   82/01/04   82004     820104
STEVENS          PRODUCTION   80/06/02   80154     800602
```

HEXBYT: Converting a Number to a Character

The HEXBYT subroutine allows you to obtain the ASCII or EBCDIC character equivalent of a decimal integer value. This subroutine returns a single alphanumeric character in the ASCII or EBCDIC character set. You can use this subroutine to produce characters that are not on your keyboard, similar to the CTRAN subroutine.

Note: The display of special characters depends upon your software and hardware; not all special characters may display. Printable EBCDIC characters and their integer equivalents are listed in Appendix E, *Character Charts*.

Available on: All platforms.

Related functions and subroutines:

CTRAN

Syntax How to Convert a Number to a Character

The syntax is

```
HEXBYT(input, output)
```

where:

input

Numeric

Is the decimal value to be translated to a single character. A value greater than 255 is treated as the remainder of (*input*/256).

output

Alphanumeric

Is the resulting alphanumeric character.

Example Report Request Inserting Braces

The following request displays the names of employees and their salaries. The names of employees earning less than $12,000 a year are enclosed in braces. The braces are produced by the HEXBYT subroutine. The integer equivalent for the left brace is 192; for the right, 208.

```
DEFINE FILE EMPLOYEE
BRACE/A17 = HEXBYT(192, 'A1') | LAST_NAME | HEXBYT(208, 'A1');
BNAME/A17 = IF CURR_SAL LT 12000 THEN BRACE
   ELSE LAST_NAME;
END
TABLE FILE EMPLOYEE
PRINT BNAME CURR_SAL BY EMP_ID
END
```

The resulting report is:

```
PAGE     1


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

Note: If you are using the Hot Screen facility, some unusual characters cannot be displayed. If Hot Screen does not support the character you chose, enter the FOCUS command

```
SET SCREEN = OFF
RETYPE
```

and redisplay the report which will appear as regular terminal output. If your terminal can display the character, the character will appear. The display of special characters depends upon your software and hardware; not all special characters may display.

HHMMSS: Returning the Current Time

The HHMMSS subroutine retrieves the current time from the system. It returns the time as an eight-character string with embedded periods separating the hours, minutes, and seconds.

Note:

- &TOD returns the current time of day.
- Compiled MODIFY procedures must use the HHMMSS subroutine to obtain the time; they cannot use the &TOD variable. The &TOD variable is made current only when you execute a MODIFY, SCAN, or FSCAN procedure.

Available on: All platforms.

Syntax How to Retrieve the Current Time

The syntax is

```
HHMMSS(outfield)
```

where:

outfield

Alphanumeric

Is the name of the field to which the time (in HH.MM.SS format) is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Displaying Current Time

The following example retrieves the current time and displays it in a report footing:

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT CURR_SAL AND COMPUTE
NOWTIME/A8 = HHMMSS(NOWTIME); NOPRINT
BY LAST_NAME BY FIRST_NAME
FOOTING
"SALARY REPORT RUN AT TIME <NOWTIME"
END
```

The example produces the following report:

```
PAGE     1


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

SALARY REPORT RUN AT TIME 13.44.30
```

IMOD, FMOD, and DMOD: Calculating the Remainder From a Division

The MOD subroutines calculate the remainder from a division. There are three MOD subroutines:

- IMOD returns the remainder as an integer.
- FMOD returns the remainder as a floating-point number.
- DMOD returns the remainder as a decimal number.

The three subroutines use the formula:

```
remainder = dividend - INT(dividend/divisor) * divisor
```

Available on: All platforms.

Related functions and subroutines:

INT

Syntax How to Calculate the Remainder From a Division

The syntax is

subroutine(*dividend*, *divisor*, *outfield*)

where:

subroutine

Is one of the following:

IMOD returns the remainder as an integer.

FMOD returns the remainder as a floating-point number.

DMOD returns the remainder as a decimal number.

dividend

Numeric

Is the dividend.

divisor

Numeric

Is the divisor.

outfield

Numeric

Is the name of the field to which the remainder is returned. Remember that the subroutine name (IMOD, FMOD, or DMOD) determines the format. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Extracting Last Three Digits From Account Number

The following example extracts the last three digits from the employee bank account numbers by dividing by 1000 and finding the remainder:

```
TABLE FILE EMPLOYEE
PRINT ACCTNUMBER AND
COMPUTE LAST3_ACCT/I3L = IMOD(ACCTNUMBER, 1000, LAST3_ACCT);
BY LAST_NAME
BY FIRST_NAME
WHERE ACCTNUMBER NE 000000000;
END
```

The example produces the following report:

```
PAGE     1

LAST_NAME        FIRST_NAME  ACCTNUMBER  LAST3_ACCT
---------        ----------  ----------  ----------
BLACKWOOD        ROSEMARIE    122850108         108
CROSS            BARBARA      163800144         144
GREENSPAN        MARY         150150302         302
IRVING           JOAN         819000702         702
JONES            DIANE        040950036         036
MCCOY            JOHN         109200096         096
MCKNIGHT         ROGER        136500120         120
ROMANS           ANTHONY      095550084         084
SMITH            MARY         027300024         024
                 RICHARD      054600048         048
STEVENS          ALFRED       013650102         102
```

INT: Finding the Greatest Integer

The INT function returns the integer part of its argument.

Available on: All platforms.

Related functions and subroutines:

IMOD, FMOD, and DMOD

Syntax How to Calculate the Greatest Integer

The syntax is

```
INT(argument)
```

where:

argument

Numeric

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

Example Report Request Calculating the Greatest Integer in RETAIL_COST

The following example calculates the greatest integer in the RETAIL_COST field:

```
TABLE FILE SALES
PRINT RETAIL_PRICE AND
COMPUTE INT_RETAIL_PRICE/D12.2 = INT(RETAIL_PRICE);
BY PROD_CODE
WHERE PROD_CODE CONTAINS 'B';
END
```

The example produces the following report:

```
PAGE     1


PROD_CODE  RETAIL_PRICE  INT_RETAIL_PRICE
---------  ------------  ----------------
B10                $.95               .00
                   $.85               .00
                   $.99               .00
B12               $1.29              1.00
                  $1.49              1.00
B17               $1.89              1.00
                  $1.89              1.00
B20               $1.99              1.00
                  $2.09              2.00
```

ITONUM: Converting Large Binary Integers to Double-Precision

The ITONUM subroutine converts large binary integers in non-FOCUS files to double-precision format. Some programming languages and some non-FOCUS data storage systems use large binary integer formats. Large binary integers (more than 4 bytes in length) are not supported in the Master File syntax and, therefore, require conversion to double-precision format.

The ITONUM subroutine processes a large byte binary format input string and converts it to a double-precision number. The user specifies how many of the rightmost bytes in the input string are significant. The output of ITONUM is an 8-byte double-precision field.

Available on: MVS, VM/CMS, WebFOCUS.

Related functions and subroutines:

ITOPACK

Syntax How to Convert Large Binary Integers to Double-Precision

The syntax is

```
ITONUM(sigbytes, infield, outfield)
```

where:

sigbytes

Numeric

Is the maximum number of bytes in the 8-byte binary input field that have significant numeric data, including the binary sign. Valid values are:

`5` The left-most 3 bytes are ignored.

`6` The left-most 2 bytes are ignored.

`7` The left-most byte is ignored.

infield

A8

Is the field that contains the binary number. Both the USAGE and ACTUAL formats must be A8.

outfield

Numeric

Is the name of the field that contains the double-precision number. This argument can also be the format of the output value, enclosed in single quotation marks. The format must be specified as Dn or Dn.d.

Example Converting a Large Binary Integer to Double-Precision

Suppose a binary number in an external file has the following COBOL format:

```
PIC 9(8)V9(4) COMP
```

It is defined in the EUROCAR Master File as a field called BINARYFLD. Its field formats are USAGE=A8 and ACTUAL=A8, since its length is greater than 4 bytes.

The field can be converted to a double-precision number using the following request:

```
DEFINE FILE EUROCAR
MYFLD/D12.2 = ITONUM(6, BINARYFLD, MYFLD);
END
TABLE FILE EUROCAR
PRINT MYFLD BY CAR
END
```

ITOPACK: Converting Large Binary Integers to Packed-Decimal Format

The ITOPACK subroutine converts large binary integers in non-FOCUS files to packed-decimal format. Some programming languages and some non-FOCUS data storage systems use double-word binary integer formats. These are similar to the single-word binary integers used by FOCUS, but they allow larger numbers. Large binary integers (more than 4 bytes in length) are not supported in the Master File syntax and, therefore, require conversion to packed format.

The ITOPACK subroutine processes an 8-byte binary format input string and converts it to a packed number. The user specifies how many of the rightmost bytes in the input string are significant. The output of ITOPACK is an 8-byte packed field of up to 15 significant numeric positions (for example, P15 or P16.d).

Available on: MVS, VM/CMS, WebFOCUS.

Related functions and subroutines:

ITONUM

Syntax How to Convert Large Binary Integers to Packed-Decimal Format

The syntax is

```
ITOPACK(sigbytes, infield, outfield)
```

where:

sigbytes

Numeric

Is the maximum number of bytes in the 8-byte binary input field that have significant numeric data, including the binary sign. Valid values are:

5	Up to 11 significant positions (the first 3 bytes are ignored).
6	Up to 14 significant positions (the first 2 bytes are ignored).
7	Up to 15 significant positions (the first byte is ignored).

infield

A8

Is the field that contains the binary number. Both the USAGE and ACTUAL formats must be A8.

outfield

Numeric

Is the name of the field that contains the packed number. This argument can also be the format of the output value, enclosed in single quotation marks. The format must be specified as Pn or Pn.d.

Note: The only restriction is that for a field defined as ‘PIC 9(15) COMP’ or the equivalent (15 significant digits), the maximum number that can be translated is 167,744,242,712,576.

Example Converting a Large Binary Integer to Packed-Decimal Format

Suppose a binary number in an external file has the following COBOL format:

```
PIC 9(8)V9(4) COMP
```

It is defined to FOCUS in the EUROCAR Master File as a field called BINARYFLD. Its field formats are USAGE=A8 and ACTUAL=A8, since its length is greater than 4 bytes.

The field can be converted to a packed number using the following request:

```
DEFINE FILE EUROCAR
PACKFLD/P14.4 = ITOPACK(6, BINARYFLD, PACKFLD);
END
TABLE FILE EUROCAR
PRINT PACKFLD BY CAR
END
```

ITOZ: Converting to Zoned Format

The ITOZ subroutine converts numbers in numeric format to zoned format. Although FOCUS does not process zoned numbers, FOCUS requests can write zoned fields to extract files for use by external programs.

Available on: MVS, VM/CMS, OpenVMS, WebFOCUS.

Syntax How to Convert to Zoned Format

The syntax is

```
ITOZ(outlength, number, outfield)
```

where:

outlength

Numeric

Is the length of the zoned number in bytes, up to 15 bytes. The last byte includes the sign.

number

Numeric

Is the number to be converted or the field that contains the number. The number is truncated to an integer before it is converted.

outfield

Alphanumeric

Is the name of the field that contains the zoned number. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Converting to Zoned Format

The following request prepares an extract file containing employee IDs and salaries in zoned format for a COBOL program. The request is:

```
DEFINE FILE EMPLOYEE
ZONE_SAL/A8 = ITOZ(8, CURR_SAL, ZONE_SAL);
END
TABLE FILE EMPLOYEE
PRINT ZONE_SAL BY EMP_ID
ON TABLE SAVE AS SALARIES
END
```

The resulting extract file is:

```
NUMBER OF RECORDS IN TABLE=       12  LINES=     12


EBCDIC  RECORD NAMED  SALARIES
FIELDNAME                         ALIAS         FORMAT        LENGTH

EMP_ID                            EID           A9               9
ZONE_SAL                                        A8               8

TOTAL                                                           17
DCB USED WITH FILE SALARIES IS DCB=(RECFM=FB,LRECL=00017,BLKSIZE=00340)
>
```

JULDAT: Converting From Gregorian to Julian Format

The JULDAT subroutine converts dates from year-month-day format to Julian (year-day) format. Dates in Julian format are 5-digit numbers. The first two or four digits are the year, the last three digits are the number of the day counting from January 1. For example, January 1, 1987 in Julian format is 87001, and December 31, 1987 is 87365.

For century display, dates in Julian format are 7-digit numbers: the first four digits are the century; the last three digits are the number of the day counting from January 1. For example, January 1, 2001 in Julian format is 2001001.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

Available on: All platforms.

Related functions and subroutines:

GREGDT

Syntax How to Convert a Gregorian Date to a Julian Date

The syntax is

```
JULDAT(indate, outfield)
```

where:

indate

Numeric

Is the date or field containing the date in year-month-day format (YMD or YYMD).

outfield

Integer

Is the field to which the Julian date is returned. This argument can also be the format of the output value, enclosed in single quotation marks (I5 or I7). For Maintain, specify the field name.

Example Report Request Converting Gregorian Date to Julian Date

The following example prints employee names, department assignments, and hire dates, in both year-month-day and Julian formats:

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT HIRE_DATE
AND COMPUTE
JULIAN_DATE/I5 = JULDAT(HIRE_DATE, JULIAN_DATE);
BY LAST_NAME BY FIRST_NAME
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME  DEPARTMENT  HIRE_DATE  JULIAN_DATE
---------        ----------  ----------  ---------  -----------
BANNING          JOHN        PRODUCTION   82/08/01        82213
BLACKWOOD        ROSEMARIE   MIS          82/04/01        82091
CROSS            BARBARA     MIS          81/11/02        81306
GREENSPAN        MARY        MIS          82/04/01        82091
IRVING           JOAN        PRODUCTION   82/01/04        82004
JONES            DIANE       MIS          82/05/01        82121
MCCOY            JOHN        MIS          81/07/01        81182
MCKNIGHT         ROGER       PRODUCTION   82/02/02        82033
ROMANS           ANTHONY     PRODUCTION   82/07/01        82182
SMITH            MARY        MIS          81/07/01        81182
                 RICHARD     PRODUCTION   82/01/04        82004
STEVENS          ALFRED      PRODUCTION   80/06/02        80154
```

LAST: Retrieving the Preceding Value

The LAST function retrieves the preceding value selected for a field.

Available on: All platforms.

Syntax How to Retrieve the Preceding Value

The syntax is

```
LAST fieldname
```

where:

fieldname

Alphanumeric or Numeric

Is the field name.

The effect of the keyword LAST depends on whether it appears in a DEFINE or COMPUTE. In a DEFINE, the LAST value is that of the previous record retrieved from the file before sorting takes place. In a COMPUTE, the LAST value is that of the record in the previous line in the report.

LAST cannot be used with -SET in Dialogue Manager.

Example Report Request Displaying Running Total of Current Salaries by Department

The following example produces a running total of the CURR_SAL field within departments. It uses LAST to determine whether the previously retrieved value of DEPARTMENT equals the current value. If the values are equal, CURR_SAL is added to RUN_TOT. If the values are different, the department has changed and RUN_TOT starts with the value of the first CURR_SAL in the new department.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME CURR_SAL AND COMPUTE
RUN_TOT/D12.2M = IF DEPARTMENT EQ LAST DEPARTMENT THEN
                (RUN_TOT + CURR_SAL) ELSE CURR_SAL ;
AS 'RUNNING,TOTAL,SALARY'
BY DEPARTMENT SKIP-LINE
END
```

The example produces the following report:

```
PAGE     1

                                                      RUNNING
                                                      TOTAL
DEPARTMENT  LAST_NAME               CURR_SAL          SALARY
----------  ---------               --------          -------

MIS         SMITH                 $13,200.00       $13,200.00
            JONES                 $18,480.00       $31,680.00
            MCCOY                 $18,480.00       $50,160.00
            BLACKWOOD             $21,780.00       $71,940.00
            GREENSPAN              $9,000.00       $80,940.00
            CROSS                 $27,062.00      $108,002.00

PRODUCTION  STEVENS               $11,000.00       $11,000.00
            SMITH                  $9,500.00       $20,500.00
            BANNING               $29,700.00       $50,200.00
            IRVING                $26,862.00       $77,062.00
            ROMANS                $21,120.00       $98,182.00
            MCKNIGHT              $16,100.00      $114,282.00
```

LCWORD: Converting Letters in a Word to Mixed Case

The LCWORD subroutine converts the letters in the given string to mixed case. The subroutine converts to lowercase every alphanumeric character except:

- The first letter of each new word.
- The first letter after a single or double quotation mark. For example, O'CONNOR is converted to O'Connor and JACK'S to Jack'S (not Jack's).

The rest of the word is converted to lowercase. The result is a word with an initial uppercase character followed by lowercase characters.

If the subroutine encounters a number in the string, the subroutine treats it as an uppercase character and continues to convert the following alphabetic characters to lowercase.

Available on: MVS, VM/CMS, WebFOCUS, Windows.

Related functions and subroutines:

- LOCASE
- UPCASE

Syntax How to Convert Letters to Mixed Case

The syntax is

```
LCWORD(inlength, infield, outfield)
```

where:

inlength

Integer

Is the length of the input field.

infield

Alphanumeric

Is the name of the input field or the input string enclosed in single quotation marks.

outfield

Alphanumeric

Is the name of the output field. The length of the outfield must be greater than or equal to the length of the infield.

Example Report Request Converting LAST_NAME to Mixed Case

The following example converts LAST_NAME values, which are all uppercase, to mixed case:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
MIXED_CASE/A15 = LCWORD(15, LAST_NAME, MIXED_CASE) ;
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        MIXED_CASE
---------        ----------
STEVENS          Stevens
SMITH            Smith
JONES            Jones
SMITH            Smith
BANNING          Banning
IRVING           Irving
ROMANS           Romans
MCCOY            Mccoy
BLACKWOOD        Blackwood
MCKNIGHT         Mcknight
GREENSPAN        Greenspan
CROSS            Cross
```

LJUST: Left-justifying a String

The LJUST subroutine left-justifies a character string within a field. All leading spaces become trailing spaces. The LJUST subroutine is helpful in left-justifying character strings previously right-justified or centered.

Available on: All platforms.

Related functions and subroutines:

- CTRFLD
- RJUST

Note: Use of LJUST in a styled report (StyleSheets feature) generally negates the effect of this feature unless the item is also styled as a centered element.

Syntax How to Left-justify a String

The syntax is

```
LJUST(inlength, infield, outfield)
```

where:

inlength

Integer

Is the length of infield and outfield.

infield

Alphanumeric

Is the name of the data field to be left-justified or the input string enclosed in single quotation marks.

outfield

Alphanumeric

Is the name of the field to which the output is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Left-justifying a Formerly Numeric Field

The following example converts current salaries from numeric to alphanumeric format using the FTOA subroutine. It then left-justifies the resulting alphanumeric values.

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND COMPUTE
SAL_STRING/A12 = FTOA(CURR_SAL, '(D8.2M)', SAL_STRING);
LEFT_SAL/A12 = LJUST(12, SAL_STRING, LEFT_SAL);
BY LAST_NAME
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME  SAL_STRING    LEFT_SAL
---------        ----------  ----------    --------
BANNING          JOHN        $29,700.00    $29,700.00
BLACKWOOD        ROSEMARIE   $21,780.00    $21,780.00
CROSS            BARBARA     $27,062.00    $27,062.00
GREENSPAN        MARY         $9,000.00    $9,000.00
IRVING           JOAN        $26,862.00    $26,862.00
JONES            DIANE       $18,480.00    $18,480.00
MCCOY            JOHN        $18,480.00    $18,480.00
MCKNIGHT         ROGER       $16,100.00    $16,100.00
ROMANS           ANTHONY     $21,120.00    $21,120.00
SMITH            MARY        $13,200.00    $13,200.00
                 RICHARD      $9,500.00    $9,500.00
STEVENS          ALFRED      $11,000.00    $11,000.00
```

LOCASE: Converting Text to Lowercase

The LOCASE subroutine converts the alphabetical text in a field to lowercase.

This is useful for converting input fields from FIDEL CRTFORMs and from non-FOCUS applications to lowercase.

Available on: All platforms.

Related functions and subroutines:

- LCWORD
- UPCASE

Syntax How to Convert Text to Lowercase

The syntax is

```
LOCASE(inlength, infield, outfield)
```

where:

inlength

Integer

Is the length of the input and output fields. It must be greater than 0. The length, in characters, must be equal for both arguments; otherwise, an error occurs.

infield

Alphanumeric

Is the name of the field to convert or the input string enclosed in single quotation marks.

outfield

Alphanumeric

Is the name of the field in which to store the converted text. This can be the same as the infield. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Converting LAST_NAME to Lowercase

The following example converts LAST_NAME values, which are all uppercase, to lowercase:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
LOWER_NAME/A15 = LOCASE(15, LAST_NAME, LOWER_NAME);
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        LOWER_NAME
---------        ----------
STEVENS          stevens
SMITH            smith
JONES            jones
SMITH            smith
BANNING          banning
IRVING           irving
ROMANS           romans
MCCOY            mccoy
BLACKWOOD        blackwood
MCKNIGHT         mcknight
GREENSPAN        greenspan
CROSS            cross
```

LOG: Calculating the Natural Logarithm

The LOG function returns the natural logarithm of its argument.

Available on: All platforms.

Related functions and subroutines:

EXP

Syntax How to Calculate the Natural Logarithm

The syntax is

```
LOG(argument)
```

where:

argument

Numeric

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation. If you enter an argument less than or equal to 0, LOG returns 0.

Example Report Request Calculating Natural Logarithm of Gross Sales

The following example calculates the log of the gross sales (determined by multiplying the number of products sold (UNIT_SOLD) by the retail price (RETAIL_PRICE):

```
TABLE FILE SALES
PRINT UNIT_SOLD AND RETAIL_PRICE AND
COMPUTE GROSS/D12.2M = UNIT_SOLD * RETAIL_PRICE; AND
COMPUTE LOG_GROSS/D12.2 = LOG( GROSS );
BY PROD_CODE
WHERE DATE LE '1017';
END
```

The example produces the following report:

```
PAGE       1


PROD_CODE     UNIT_SOLD     RETAIL_PRICE     GROSS     LOG_GROSS
---------     ---------     ------------     -----     ---------
B10                  30             $.85    $25.50          3.24
B17                  20            $1.89    $37.80          3.63
B20                  15            $1.99    $29.85          3.40
C17                  12            $2.09    $25.08          3.22
D12                  20            $2.09    $41.80          3.73
E1                   30             $.89    $26.70          3.28
E3                   35            $1.09    $38.15          3.64
```

MAX and MIN: Finding the Maximum or Minimum Value

The MAX and MIN functions return either the maximum or minimum value (respectively) from a list of arguments.

Available on: All platforms.

Syntax How to Find the Maximum or Minimum Value

The syntax for MAX is

```
MAX(argument1, argument2, ...)
```

and the syntax for MIN is

```
MIN(argument1, argument2, ...)
```

where:

argument1, *argument2*

Numeric

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

Example Report Request Determining Minimum Loss Due to Returns and Damaged Items

The following example finds the minimum value from the RETURNS and DAMAGED fields:

```
TABLE FILE SALES
PRINT RETURNS AND DAMAGED AND
COMPUTE MIN_LOSS/I3 = MIN(RETURNS, DAMAGED);
BY PROD_CODE
WHERE PROD_CODE CONTAINS 'B';
END
```

This example produces the following report:

```
PAGE     1


PROD_CODE  RETURNS  DAMAGED  MIN_LOSS
---------  -------  -------  --------
B10             10        6         6
                 2        3         2
                 1        1         1
B12              3        3         3
                 1        0         0
B17              2        1         1
                 2        1         1
B20              0        1         0
                 1        1         1
```

MVSDYNAM: Passing a DYNAM Command to the Command Processor

The MVSDYNAM subroutine transfers a specified FOCUS DYNAM command to the DYNAM command processor. A zero (0) return code indicates successful processing; non-zero codes indicate failure. This is useful in compiled MODIFY procedures after the CASE AT START statement to pass allocation statements to the processor.

Available on: MVS.

Syntax How to Pass a DYNAM Command to the Command Processor

The syntax is

```
MVSDYNAM(command, length, rc)
```

where:

command

Alphanumeric

Is the DYNAM command, enclosed in single quotation marks, or a field or variable that contains the command. The subroutine converts lowercase input to uppercase.

length

Numeric

Is the command length from 1 to 256 characters long.

rc

I4

Is the name of the field that contains the return code. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

MVSDYNAM returns one of three possible types of codes:

`0`
: The DYNAM command transferred and successfully executed.

`> 0, positive number`
: FOCUS error number corresponding to a FOCUS error.

`< 0, negative number`
: FOCUS error number corresponding to DYNAM failure (from the SVC).

Example Executing the DYNAM FREE Command

In this MODIFY procedure, the MVSDYNAM subroutine transfers the DYNAM FREE command to the processor. Query commands display the results before and after the DYNAM FREE command is specified. The successful return code of zero (0) is stored in the RES field.

```
-* THE RESULT OF ? TSO DDNAME CAR WILL BE BLANK AFTER ENTERING
-* 'FREE FILE CAR' AS YOUR COMMAND
DYNAM ALLOC FILE CAR DS USER1.CAR.FOCUS SHR REUSE
? TSO DDNAME CAR
-RUN
-PROMPT &XX.ENTER A SPACE TO CONTINUE.
MODIFY FILE CAR
COMPUTE LINE/A60=;
        RES/I4 = 0;
CRTFORM
" ENTER DYNAM COMMAND BELOW:"
" <LINE>"
COMPUTE
RES = MVSDYNAM(LINE, 60, RES);
GOTO DISPLAY

 CASE DISPLAY
 CRTFORM LINE 1
" THE RESULT OF DYNAM WAS <D.RES"
GOTO EXIT
ENDCASE
DATA
END
? TSO DDNAME CAR
```

The first query command displays the allocation that results from the DYNAM ALLOCATE command. Type one space and press the Enter key to continue.

```
DDNAME      =  CAR
DSNAME      =  USER1.CAR.FOCUS
DISP        =  SHR
DEVICE      =  DISK
VOLSER      =  USERMN
DSORG       =  PS
RECFM       =  F
SECONDARY   =   100
ALLOCATION  =  BLOCKS
BLKSIZE     =      4096
LRECL       =      4096
TRKTOT      =         8
EXTENTSUSED =         1
BLKSPERTRK  =        12
TRKSPERCYL  =        15
CYLSPERDISK =      2227
BLKSWRITTEN =        96
FOCUSPAGES  =         8
ENTER A SPACE TO CONTINUE  >
```

Then, enter the DYNAM FREE command. (The DYNAM keyword is assumed.)

```
ENTER DYNAM COMMAND BELOW:
 free file car
```

The subroutine successfully transfers the DYNAM FREE command to the processor and the return code displays. Press the Enter key to continue.

```
THE RESULT OF DYNAM WAS     0
```

Then, the second query command indicates that the allocation has been freed.

```
DDNAME      =  CAR
DSNAME      =
DISP        =
DEVICE      =
VOLSER      =
DSORG       =
RECFM       =
SECONDARY   =  ****
ALLOCATION  =
BLKSIZE     =         0
LRECL       =         0
TRKTOT      =         0
EXTENTSUSED =         0
BLKSPERTRK  =         0
TRKSPERCYL  =         0
CYLSPERDISK =         0
BLKSWRITTEN =         0
>
```

OVRLAY: Overlaying a Substring Within a String

The OVRLAY subroutine overlays a substring on another character string. When specified in MODIFY procedures, the subroutine enables you to edit a part of an alphanumeric field without replacing the field entirely.

Available on: All platforms.

Related functions and subroutines:

- POSIT
- SUBSTR

Syntax How to Overlay a Substring

The syntax is

```
OVRLAY(base, baselen, substring, sublen, position, outfield)
```

where:

base

Alphanumeric

Is the character string to be overlaid.

baselen

Integer

Is the length of the base and outfield strings. If this argument is less than or equal to 0, unpredictable results occur.

substring

Alphanumeric

Is the substring to overlay the base string.

sublen

Integer

Is the length of the substring. If this argument is less than or equal to 0, the subroutine returns spaces.

position

Integer

Is the position in the base string where the overlay is to begin. If this argument is less than or equal to 0, the subroutine returns spaces. If the argument is larger than baselen, the subroutine returns the base string.

outfield

Alphanumeric

Is the name of the field to which the overlaid string is returned. If the overlaid string is longer than the output field, the string is truncated to fit the field. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Replacing Last Three Characters of EMP_ID

The following example replaces the last three characters of EMP_ID (starting at the 7th position) with the three-character job code found in CURR_JOBCODE, creating a new security identification code:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND COMPUTE
NEW_ID/A9 = OVRLAY(EMP_ID, 9, CURR_JOBCODE, 3, 7, NEW_ID);
END
```

The example produces the following report:

```
PAGE     1


EMP_ID     NEW_ID
------     ------
071382660  071382A07
112847612  112847B14
117593129  117593B03
119265415  119265A01
119329144  119329A17
123764317  123764A15
126724188  126724B04
219984371  219984B02
326179357  326179B04
451123478  451123B02
543729165  543729A07
818692173  818692A17
```

Example MODIFY Request Using OVRLAY

This MODIFY procedure prompts for input using a CRTFORM screen and updates first names in the EMPLOYEE data source. The CRTFORM LOWER option enables you to update the names in lowercase, but the procedure ensures that the first letter of each name is capitalized. The procedure is:

```
MODIFY FILE EMPLOYEE
CRTFORM LOWER
  "ENTER EMPLOYEE'S ID: <EMP_ID"
  "ENTER FIRST_NAME IN LOWER CASE: <FIRST_NAME"
MATCH EMP_ID
 ON NOMATCH REJECT
 ON MATCH COMPUTE
  F_UP/A1  = UPCASE (1, FIRST_NAME,'A1');
  FIRST_NAME/A10 = OVRLAY (FIRST_NAME, 10, F_UP,
                  1, 1, 'A10');
  ON MATCH TYPE "CHANGING FIRST NAME TO <FIRST_NAME "
  ON MATCH UPDATE FIRST_NAME
DATA
END
```

The COMPUTE statement invokes two subroutines:

- The UPCASE subroutine extracts the first letter and converts it to uppercase.
- The OVRLAY subroutine replaces the present first letter in the name with the uppercase initial.

A sample execution is:

```
ENTER EMPLOYEE'S ID:  071382660
ENTER FIRST_NAME IN LOWER CASE:  alfred
```

The procedure processes as:

1. The procedure prompts you from a CRTFORM screen for an employee ID and a first name. You type the following data and press the Enter key:

   ```
   EMPLOYEE'S ID: 071382660
   FIRST NAME:    alfred
   ```

2. The procedure searches the data source for the ID 071382660. If it finds the ID, it continues processing the transaction. In this case, the ID exists and belongs to Alfred Stevens.

3. The UPCASE subroutine extracts the letter a from alfred and converts it to the letter A.

4. The OVRLAY subroutine overlays the letter A on alfred. The first name is now Alfred.

```
ENTER EMPLOYEE'S ID:
ENTER FIRST_NAME IN LOWER CASE:

CHANGING FIRST NAME TO Alfred
```

5. The procedure updates the first name in the data source.

6. When you exit the procedure with PF3, the FOCUS transaction message indicates that one update occurred.

```
TRANSACTIONS:         TOTAL =     1  ACCEPTED=     1  REJECTED=     0
SEGMENTS:             INPUT =     0  UPDATED =     1  DELETED =     0
```

PARAG: Dividing Text Into Smaller Lines

The PARAG subroutine divides lines of text into smaller lines by marking them off with a delimiter character. The GETTOK subroutine can then place the smaller lines, called sublines, into different fields.

The PARAG subroutine works by scanning a specific number of characters from the beginning of the line and replacing the last space in this group with a delimiter. It then scans the next group of characters starting from the delimiter and replaces the last space in this group with a second delimiter. It repeats this process until the end of the line. Each group of characters marked off by the delimiter becomes a subline.

If the subroutine finds no spaces in the group it scans, it replaces the next character after the group with the delimiter. Therefore, be sure that no word of text is longer than the number of characters scanned by the subroutine (the maximum subline length).

Available on: All platforms.

Related functions and subroutines:

GETTOK

Syntax How to Divide Text Into Smaller Lines

The syntax is

```
PARAG(inlen, infield, 'delim', subsize, outfield)
```

where:

inlen

Integer

Is the length of input string and the outfield.

infield

Alphanumeric

Is the input string.

delim

Alphanumeric

Is the delimiter character. Choose a character that does not appear in the text.

subsize

Integer

Is the maximum length of the subline.

outfield

Alphanumeric

Is the name of the field to which the delimited text is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Note: If the input lines of text are roughly equal in length, you can keep the sublines equal by specifying a subline length that evenly divides into the length of the text lines. For example, if you are dividing text lines 120 characters long, you can divide each of them into two sublines 60 characters long, three sublines 40 characters long, and so on. This enables you to print lines of text in paragraph form.

However, if you divide the lines evenly, you may create more sublines than you intend. For example, suppose you divide 120-character text lines into two lines of 60 characters maximum length. One line is divided so that the first subline is 50 characters long and the second is 55. This leaves room for a third subline 15 characters long.

To correct this, insert a space (using weak concatenation) at the beginning of the extra subline, then append this subline (using strong concatenation) to the end of the one before it.

Example Report Request Dividing Address Line Into Smaller Lines

The following example divides an address line into smaller lines using commas as delimiters:

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN2 AND COMPUTE
PARA_ADDR/A20 = PARAG(20, ADDRESS_LN2, ',', 10, PARA_ADDR);
BY LAST_NAME
WHERE TYPE EQ 'HSM'
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        ADDRESS_LN2           PARA_ADDR
---------        -----------           ---------
BANNING          APT 4C                APT 4C    ,
CROSS            147-15 NORTHERN BLD   147-15,NORTHERN,BLD
GREENSPAN        13 LINDEN AVE.        13,LINDEN,AVE.
IRVING           123 E 32 ST.          123 E 32,ST.       ,
JONES            235 MURRAY HIL PKWY   235 MURRAY,HIL PKWY
MCKNIGHT         117 HARRISON AVE.     117,HARRISON,AVE.
ROMANS           271 PRESIDENT ST.     271,PRESIDENT,ST.
SMITH            136 E 161 ST.         136 E 161,ST.   ,
```

PCKOUT: Writing Packed Numbers of Different Lengths

The PCKOUT subroutine enables requests to write packed numbers of different lengths to extract files (HOLD and SAVE files). When a request saves packed fields in extract files, it writes them as 8- or 16-byte fields regardless of their format specifications. With the PCKOUT subroutine, you can vary their lengths between 1 to 16 bytes.

Available on: MVS, VM/CMS, UNIX, OpenVMS, WebFOCUS.

Related functions and subroutines:

- CHKPCK
- ITOPACK

Syntax How to Write Packed Numbers of Different Lengths

The syntax is

```
PCKOUT(infield, outlength, outfield)
```

where:

infield

Numeric

Is the input field that contains the values. The field can be packed, integer, floating-point or double-precision. If the field is not integer, its values are rounded to the nearest integer.

outlength

Numeric

Is the output field length from 1 to 16 bytes.

outfield

Alphanumeric

Is the name of the output field written to the extract file. The subroutine returns the field as alphanumeric although it contains packed data. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Writing Packed Numbers of Different Lengths

This report request writes names, salaries, and dates of hire to a SAVE file. The salaries from the CURR_SAL field (USAGE=D12.2M) are converted and written to the 5-byte packed field SHORT_SAL:

```
DEFINE FILE EMPLOYEE
SHORT_SAL/A5 = PCKOUT(CURR_SAL, 5, SHORT_SAL);
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME SHORT_SAL HIRE_DATE
ON TABLE SAVE
END
```

After FOCUS creates the SAVE file, the FOCUS message returns the fields and their lengths:

```
>
 NUMBER OF RECORDS IN TABLE=       12  LINES=     12


 EBCDIC  RECORD NAMED  SAVE
 FIELDNAME                         ALIAS         FORMAT        LENGTH

 LAST_NAME                         LN            A15            15
 SHORT_SAL                                       A5              5
 HIRE_DATE                         HDT           I6YMD           6

 TOTAL                                                          26
 DCB USED WITH FILE SAVE     IS DCB=(RECFM=FB,LRECL=00026,BLKSIZE=00520)
>
```

POSIT: Finding Substring Position

The POSIT subroutine finds the starting positions of substrings within larger strings. For example, the position of the substring DUCT in the character string PRODUCTION is position 4.

If the substring is not in the parent string, the subroutine returns the value 0.

Available on: All platforms.

Related functions and subroutines:

OVRLAY

Syntax How to Find a Substring Position

The syntax is

`POSIT(`*`parent`*`, `*`inlength`*`, `*`substring`*`, `*`sublength`*`, `*`outfield`*`)`

where:

`parent`

Alphanumeric

Is the field containing the parent character string.

`inlength`

Integer

Is the parent field length. If this argument is less than or equal to 0, the subroutine returns 0.

substring

Alphanumeric

Is the substring whose position you wish to find, enclosed in single quotation marks, or a field that contains the string.

sublength

Integer

Is the length of *substring*. If this argument is less than or equal to 0, or if it is greater than the *inlength* argument, the subroutine returns a 0.

outfield

Integer

Is the name of the field to which the position is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Determining First Position of the Letter I in LAST_NAME

The following example displays the positions of the first capital letter I in last names:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2');
BY EMP_ID
END
```

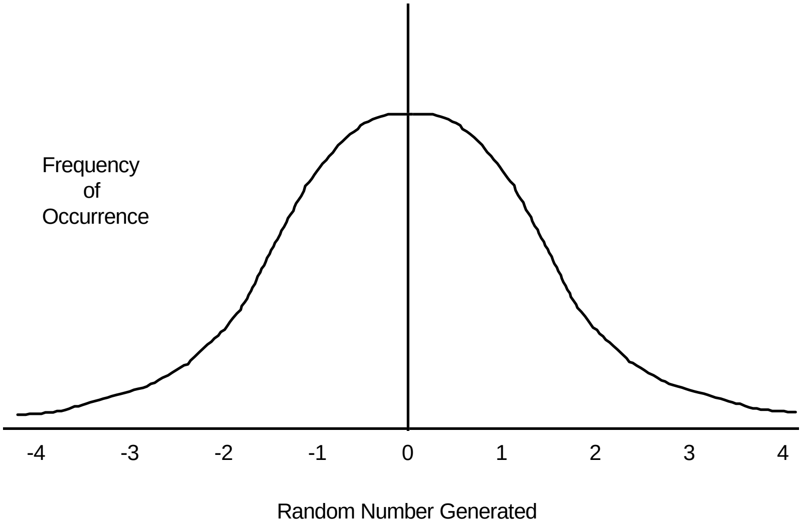
The example produces the following report:

```
PAGE     1


EMP_ID     LAST_NAME        I_IN_NAME
------     ---------        ---------
071382660  STEVENS                  0
112847612  SMITH                    3
117593129  JONES                    0
119265415  SMITH                    3
119329144  BANNING                  5
123764317  IRVING                   1
126724188  ROMANS                   0
219984371  MCCOY                    0
326179357  BLACKWOOD                0
451123478  MCKNIGHT                 5
543729165  GREENSPAN                0
818692173  CROSS                    0
```

PRDNOR, PRDUNI, RDNORM, and RDUNIF: Generating Random Numbers

The PRDNOR, PRDUNI, RDNORM, and RDUNIF subroutines generate random numbers:

- RDNORM generates double-precision random numbers that are normally distributed with an arithmetic mean of 0 and a standard deviation of 1. If you use the RDNORM subroutine to generate a large set of numbers (between 1 and 32768), it has the following properties:
  - The numbers in the set lie roughly on a bell curve, as shown in the following figure. The bell curve is highest at the 0 mark, which means that there are more numbers close to 0 than farther away.



  - The average of the set is close to 0.
  - The set can contain numbers of any size, but most of the numbers are between 3 and -3.
- PRDNOR does the same thing as RDNORM, except that the set of random numbers is *reproducible*.
- RDUNIF generates double-precision random numbers uniformly distributed between 0 and 1 (that is, any random number it generates has an equal probability of being anywhere between 0 and 1).
- PRDUNI does the same thing as RDUNIF, except that the set of random numbers is *reproducible*.

Available on: All platforms.

Syntax How to Use RDNORM and RDUNIF to Generate Random Numbers

The syntax is

subroutine(*outfield*)

where:

subroutine

Is one of the following:

RDNORM generates normally distributed random numbers with an arithmetic mean of 0 and a standard deviation of 1.

RDUNIF generates random numbers uniformly distributed between 0 and 1.

outfield

Double-precision

Is the name of the double-precision field that contains the random numbers. This argument can also be the format of the output value, enclosed in single quotation marks.

Syntax How to Use PRDNOR and PRDUNI to Generate Random Numbers

The syntax is

subroutine(*seed, outfield*)

where:

subroutine

Is one of the following:

PRDNOR generates reproducible normally distributed random numbers with an arithmetic mean of 0 and a standard deviation of 1.

PRDUNI generates reproducible random numbers uniformly distributed between 0 and 1.

seed

Numeric

Is the seed or the field that contains the seed, up to nine bytes. The seed is truncated to an integer.

outfield

Double-precision

Is the name of the field that contains the random numbers. This argument can also be the format of the output value, enclosed in single quotation marks.

Note: For the PRDUNI subroutine, CMS behavior differs from MVS behavior. In CMS, the seed number changes upon multiple executions as the subroutine is reloaded. In MVS, the subroutine is loaded once. To keep the subroutine loaded for the duration of the session, we recommend assigning the subroutine to a temporary field using a DEFINE command. The subroutine remains loaded in memory until the DEFINE is cleared.

Example Using RDNORM to Generate Random Numbers

Suppose your company is granting bonuses that range between $1 and $200, with the average bonus at $100. The company is granting bonuses on a bell curve: More employees receive a near-average bonus than receive a large or small one. You develop a request that randomly assigns bonuses on a bell curve and calculates what the cost to each department would be. The request is:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME CURR_SAL AND COMPUTE
    DIFFERENCE/D7.2M = RDNORM(DIFFERENCE) * 30;
    BONUS/D7.2M = 100 + DIFFERENCE;
BY DEPARTMENT BY EMP_ID
ON DEPARTMENT SUBTOTAL BONUS
END
```

The COMPUTE statement calculates the DIFFERENCE field, which is the difference between each bonus and the average $100. The RDNORM subroutine generates numbers mostly between 3 and -3; thus, multiplying by 30 will produce numbers ranging mostly between 90 and -90. The COMPUTE statement then adds this amount to $100 to calculate the bonus.

The request produces the following report:

```
PAGE     1


DEPARTMENT EMP_ID    LAST_NAME              CURR_SAL DIFFERENCE     BONUS
---------- ------    ---------              -------- ----------     -----
MIS        112847612 SMITH                $13,200.00    -$58.37    $41.63
           117593129 JONES                $18,480.00     $11.04   $111.04
           219984371 MCCOY                $18,480.00      $3.27   $103.27
           326179357 BLACKWOOD            $21,780.00      $5.72   $105.72
           543729165 GREENSPAN             $9,000.00     $13.49   $113.49
           818692173 CROSS                $27,062.00     $23.16   $123.16

*TOTAL DEPARTMENT MIS
                                                                  $598.31

PRODUCTION 071382660 STEVENS              $11,000.00    -$25.31    $74.69
           119265415 SMITH                 $9,500.00    -$17.72    $82.28
           119329144 BANNING              $29,700.00      $6.32   $106.32
           123764317 IRVING               $26,862.00    -$39.95    $60.05
           126724188 ROMANS               $21,120.00      $3.08   $103.08
           451123478 MCKNIGHT             $16,100.00     $77.14   $177.14

*TOTAL DEPARTMENT PRODUCTION
                                                                  $603.55

TOTAL                                                           $1,201.86
```

Example Using RDUNIF to Generate Random Numbers

Suppose your company grants 30% of its employees a 5% bonus every year. Every employee is equally eligible for the raise. You develop a request that randomly assigns bonuses to 30% of your employees and calculates what the cost to each department would be. The request is:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
CHANCE/D4.3 = RDUNIF(CHANCE);
BONUS/D12.2M   = IF CHANCE LE .30
    THEN CURR_SAL * .05 ELSE 0;
BY DEPARTMENT BY EMP_ID
ON DEPARTMENT SUBTOTAL BONUS
END
```

The request produces the following report:

```
PAGE     1


DEPARTMENT  EMP_ID            CURR_SAL  CHANCE            BONUS
----------  ------            --------  ------            -----
MIS         112847612       $13,200.00    .960              $.00
            117593129       $18,480.00    .363              $.00
            219984371       $18,480.00    .469              $.00
            326179357       $21,780.00    .837              $.00
            543729165        $9,000.00    .348              $.00
            818692173       $27,062.00    .240         $1,353.10

*TOTAL DEPARTMENT MIS
                                                       $1,353.10

PRODUCTION  071382660       $11,000.00    .036           $550.00
            119265415        $9,500.00    .691              $.00
            119329144       $29,700.00    .295         $1,485.00
            123764317       $26,862.00    .667              $.00
            126724188       $21,120.00    .169         $1,056.00
            451123478       $16,100.00    .078           $805.00

*TOTAL DEPARTMENT PRODUCTION
                                                       $3,896.00

TOTAL                                                  $5,249.10
```

Example Using PRDNOR to Generate Random Numbers

This is the same example request used for the RDNORM subroutine, except, that every time you execute it, the PRDNOR subroutine produces the same results. To change the results, change the seed, specified here as 40. The request is:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME CURR_SAL AND COMPUTE
   DIFFERENCE/D7.2M = PRDNOR(40, DIFFERENCE) * 30;
   BONUS/D7.2M = 100 + DIFFERENCE;
BY DEPARTMENT BY EMP_ID
ON DEPARTMENT SUBTOTAL BONUS
END
```

The resulting report is:

```
PAGE     1


DEPARTMENT EMP_ID    LAST_NAME             CURR_SAL DIFFERENCE     BONUS
---------- ------    ---------             -------- ----------     -----
MIS        112847612 SMITH               $13,200.00     $41.39   $141.39
           117593129 JONES               $18,480.00    -$40.96    $59.04
           219984371 MCCOY               $18,480.00      $6.39   $106.39
           326179357 BLACKWOOD           $21,780.00     $16.42   $116.42
           543729165 GREENSPAN            $9,000.00    -$22.48    $77.52
           818692173 CROSS               $27,062.00       $.24   $100.24

*TOTAL DEPARTMENT MIS
                                                                 $601.00

PRODUCTION 071382660 STEVENS             $11,000.00     -$7.70    $92.30
           119265415 SMITH                $9,500.00     $33.57   $133.57
           119329144 BANNING             $29,700.00    -$27.29    $72.71
           123764317 IRVING              $26,862.00    -$25.82    $74.18
           126724188 ROMANS              $21,120.00     -$7.58    $92.42
           451123478 MCKNIGHT            $16,100.00    -$41.07    $58.93

*TOTAL DEPARTMENT PRODUCTION
                                                                 $524.11

TOTAL                                                          $1,125.11
```

Example Using PRDUNI to Generate Random Numbers

This is the same example request used for the RDUNIF subroutine, except that every time you execute it, the PRDUNI subroutine produces the same results. To change the results, change the seed, specified here as 25. The request is:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
    CHANCE/D4.3 = PRDUNI(25, CHANCE);
    BONUS/D12.2M       = IF CHANCE LE .30
    THEN CURR_SAL * .05 ELSE 0;
BY DEPARTMENT BY EMP_ID
ON DEPARTMENT SUBTOTAL BONUS
END
```

The resulting report is:

```
PAGE     1


DEPARTMENT  EMP_ID           CURR_SAL  CHANCE            BONUS
----------  ------           --------  ------            -----
MIS         112847612      $13,200.00    .555              $.00
            117593129      $18,480.00    .026           $924.00
            219984371      $18,480.00    .330              $.00
            326179357      $21,780.00    .634              $.00
            543729165       $9,000.00    .607              $.00
            818692173      $27,062.00    .908              $.00

*TOTAL DEPARTMENT MIS
                                                        $924.00

PRODUCTION  071382660      $11,000.00    .219           $550.00
            119265415       $9,500.00    .176           $475.00
            119329144      $29,700.00    .510              $.00
            123764317      $26,862.00    .756              $.00
            126724188      $21,120.00    .844              $.00
            451123478      $16,100.00    .914              $.00

*TOTAL DEPARTMENT PRODUCTION
                                                      $1,025.00

TOTAL                                                 $1,949.00
```

RJUST: Right-justifying a String

The RJUST subroutine right-justifies a character string within a field. All trailing spaces become leading spaces. This subroutine is helpful when you display alphanumeric fields containing numbers.

Available on: All platforms.

Related functions and subroutines:

- CTRFLD
- LJUST

Note: Use of RJUST in a styled report (StyleSheets feature) generally negates the effect of this feature unless the item is also styled as a centered element.

Syntax How to Right-justify a String

The syntax is

```
RJUST(inlength, infield, outfield)
```

where:

inlength

Integer

Is the length of *infield* and *outfield*.

infield

Alphanumeric

Is the input field or string enclosed in single quotation marks.

outfield

Alphanumeric

Is the name of the field to which the output is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

To avoid justification problems, *inlength* and *infield* must be the same length.

Example Report Request That Right-justifies a Field

The following example shows last names left-justified and right-justified:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
RIGHT_NAME/A15 = RJUST(15, LAST_NAME, RIGHT_NAME);
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        RIGHT_NAME
---------        ----------
STEVENS                  STEVENS
SMITH                      SMITH
JONES                      JONES
SMITH                      SMITH
BANNING                  BANNING
IRVING                    IRVING
ROMANS                    ROMANS
MCCOY                      MCCOY
BLACKWOOD              BLACKWOOD
MCKNIGHT                MCKNIGHT
GREENSPAN              GREENSPAN
CROSS                      CROSS
```

SOUNDEX: Comparing Strings Phonetically

The SOUNDEX subroutine enables you to search for character strings phonetically without knowing how they are spelled.

SOUNDEX converts character strings to 4-character codes. The first character must be the first character in the string. The last three characters represent the next three significant sounds in the string.

Available on: All platforms.

Syntax How to Compare Strings Phonetically

The syntax is

```
SOUNDEX(inlength, infield, outfield)
```

where:

inlength

A2

Is the length of the input character string. It can be a number enclosed in single quotation marks, or a field containing the number. The number must be from 1 to 99; a number larger than 99 will cause the subroutine to return asterisks (*) as output.

infield

Alphanumeric

Is the source of the input character string. It can be the character string itself (enclosed in single quotation marks), a field name, or an amper variable.

outfield

Alphanumeric

Is either the format of the output value or the name of the temporary field to which the phonetic code is returned. When used as a format, enclose in single quotation marks.

To conduct a phonetic search, do the following:

1. Use the SOUNDEX subroutine to translate data values from the field you are searching for to their phonetic codes.
2. Use the SOUNDEX subroutine to translate your best guess target string to a phonetic code. Remember that the spelling of your target string need be only approximate; however, the first letter must be correct.
3. Use a WHERE or IF statement to compare the temporary fields created in step 1 to the temporary field created in Step 2.

Example Report Request Comparing LAST_NAME Field to Micoy

The following example creates three fields:

- The field PHON_NAME contains the phonetic code of the employee’s last name.
- The field PHON_COY contains the phonetic code of your guess, Micoy.
- The field PHON_MATCH contains YES if the phonetic code matches, NO if it does not.

The WHERE statement selects the last name that matches your best guess.

```
DEFINE FILE EMPLOYEE
PHON_NAME/A4 = SOUNDEX('15', LAST_NAME, PHON_NAME);
PHON_COY/A4 WITH LAST_NAME = SOUNDEX('15', 'MICOY', PHON_COY);
PHON_MATCH/A3 = IF PHON_NAME IS PHON_COY THEN 'YES' ELSE 'NO';
END
```

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
IF PHON_MATCH IS 'YES'
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME
---------
MCCOY
```

SQRT: Calculating the Square Root

The SQRT function calculates the square root of its argument.

Available on: All platforms.

Syntax How to Calculate the Square Root

The syntax is

SQRT(*argument*)

where:

argument

Numeric

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

Example Report Request Calculating Square Root of Car's Area

The following example calculates the square root of a car's area:

```
TABLE FILE CAR
PRINT LENGTH AND WIDTH AND
COMPUTE AREA/D12 = WIDTH * LENGTH; AND
COMPUTE SQRT_AREA/D12.2 = SQRT(AREA);
BY MODEL AS 'BMW MODEL'
WHERE CAR EQ 'BMW';
END
```

This example produces the following:

```
PAGE     1


BMW MODEL                 LENGTH   WIDTH             AREA       SQRT_AREA
---------                 ------   -----             ----       ---------
2002 2 DOOR                  176      63           11,088          105.30
2002 2 DOOR AUTO             176      63           11,088          105.30
3.0 SI 4 DOOR                195      69           13,455          116.00
3.0 SI 4 DOOR AUTO           195      69           13,455          116.00
530I 4 DOOR                  190      67           12,730          112.83
530I 4 DOOR AUTO             190      67           12,730          112.83
```

SUBSTR: Extracting a Substring

The SUBSTR subroutine extracts a substring from a large character string called a parent string.

Another way to extract substrings is to use the EDIT function. The differences are:

- The EDIT function can extract a substring from different parts of the parent string. For example, it can extract the first two characters and the last two characters of a string to form a single substring. Also, it can insert characters into a substring.
- The SUBSTR subroutine can vary the position of the substring depending on the values of other fields.

Available on: All platforms.

Related functions and subroutines:

EDIT

Syntax How to Extract a Substring

The syntax is

```
SUBSTR(inlength, parent, start, end, sublength, outfield)
```

where:

`inlength`

Integer

Is the length of the parent string.

`parent`

Alphanumeric

Is the field containing the parent string or the parent string enclosed in single quotation marks.

`start`

Integer

Is the starting position of the substring in the parent string. If this argument is less than 1, the subroutine returns spaces.

`end`

Integer

Is the ending position of the substring. If this argument is less than the *start* argument or greater than the *inlength* argument, the subroutine returns spaces.

`sublength`

Integer

Is the length of the substring (normally *end* **-** *start* + 1). If the *sublength* is longer than *end* **-** *start* +1, the substring is padded with trailing spaces. If it is shorter, the substring is truncated. This value should be the declared length of *outfield*. Only *sublength* characters will be processed.

`outfield`

Alphanumeric

Is the name of the field to which the substring is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Example Extracting Three Characters From LAST_NAME Beginning With the Letter I

The following report request uses the POSIT subroutine to determine the position of the first letter I in LAST_NAME. Then the report extracts the letter I and the next two characters.

```
TABLE FILE EMPLOYEE
PRINT
COMPUTE I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2'); AND
COMPUTE I_SUBSTR/A3 = SUBSTR(15, LAST_NAME, I_IN_NAME, I_IN_NAME+2, 3,
  I_SUBSTR);
BY LAST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        I_IN_NAME  I_SUBSTR
---------        ---------  --------
BANNING                  5  ING
IRVING                   1  IRV
MCKNIGHT                 5  IGH
ROMANS                   0
SMITH                    3  ITH
STEVENS                  0
```

Notice that since Stevens and Romans have no I in their names, SUBSTR extracts a blank string.

TODAY: Returning the Current Date

The TODAY subroutine retrieves the current date from the system. It is year 2000 compatible and is useful in a compiled MODIFY. TODAY can return a 4-digit year when you declare a DEFINE or COMPUTE field as 10 bytes. TODAY can also continue returning a 2-digit year when you declare the output format as 8 bytes.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

Available on: All platforms.

Related functions and subroutines:

HHMMSS

Syntax How to Retrieve the Current Date

The syntax is

```
TODAY(outfield)
```

where:

`outfield`

Alphanumeric

Is the name of the field to which the current date in MM/DD/YY[YY] format is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Note:

- You can retrieve the date in the same format (separated by slashes) by using the system variable &DATE. You can retrieve the date without the slashes using the system variables &YMD, &MDY, and &DMY. The system variable &DATE*fmt* retrieves the date in a specified format.
- You can remove the embedded slashes using the EDIT function.
- Compiled MODIFY procedures cannot use Dialogue Manager system variables. They must use the TODAY subroutine to obtain the date.

The TODAY subroutine always returns a date that is current. Therefore, if you are running an application late at night, you may want to use the TODAY subroutine.

Example Report Request Displaying the Current Date

The following example retrieves the current date and displays it in a report heading,

```
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:

```
PAGE    1

SALARY REPORT RUN ON DATE 03/17/1999

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.

UFMT: Converting Alphanumeric to Hexadecimal

The UFMT subroutine converts characters in alphanumeric field values to hexadecimal (HEX) representation.

This subroutine is especially useful for examining data of unknown format. As long as the length of the data is known, its content can be examined.

Available on: MVS, VM/CMS, OpenVMS, WebFOCUS.

Syntax How to Convert Alphanumeric to Hexadecimal

The syntax is

```
UFMT(infield, inlength, outfield)
```

where:

infield

Alphanumeric

Is the input field or an alphanumeric string enclosed in single quotation marks.

inlength

Numeric

Is the input field length.

outfield

Alphanumeric

Is the name of the field that contains the HEX equivalent. The format of the *outfield* argument must be alphanumeric and have a length that is twice as long as the *inlength* argument (2**inlength*). This argument can also be the format of the output value, enclosed in single quotation marks.

Example Report Request Converting JOBCODE to Hexadecimal

The following request uses the UFMT subroutine to convert the values in the JOBCODE field to their HEX representation and store them in the HEXCODE temporary field. Notice that the format of the temporary field is twice as large as the *inlength* argument:

```
DEFINE FILE JOBFILE
HEXCODE/A6 = UFMT(JOBCODE, 3, HEXCODE);
END
TABLE FILE JOBFILE
PRINT JOBCODE HEXCODE
END
```

The resulting report is:

```
PAGE     1


JOBCODE  HEXCODE
-------  -------
A01      C1F0F1
A02      C1F0F2
A07      C1F0F7
A12      C1F1F2
A14      C1F1F4
A15      C1F1F5
A16      C1F1F6
A17      C1F1F7
B01      C2F0F1
B02      C2F0F2
B03      C2F0F3
B04      C2F0F4
B14      C2F1F4
```

UPCASE: Converting Text to Uppercase

The UPCASE subroutine converts a string of characters to uppercase.

One reason you might use UPCASE is when you are sorting on a field that contains both mixed case and uppercase values. In these cases, sorting uses the ASCII or EBCDIC sorting order, which may cause unpredictable results. To obtain consistent results, define a new field with all of the values in uppercase, and sort on that.

In FIDEL, CRTFORM LOWER retains the case of entries as they were typed. You can use the UPCASE subroutine to convert entries for particular fields to uppercase.

Available on: All platforms.

Related functions and subroutines:

- LCWORD
- LOCASE
- MXCASE

Syntax How to Convert Text to Uppercase

The syntax is

```
UPCASE(length, input, output)
```

where:

length

Integer

Is the length of both the *input* and the *output* strings.

input

Alphanumeric

Is the mixed-case input string or field.

output

Alphanumeric

Is the uppercase output string or field. This argument can also be the format of the output value, enclosed in single quotation marks.

Example

Report Request Converting Mixed Case Names to Uppercase

Suppose you are sorting on a field that contains both uppercase and mixed case values. The following request against a hold file, which contains data for LAST_NAME in mixed case, shows how sorting is performed in various environments:

```
TABLE FILE HOLD
PRINT FIRST_NAME BY LAST_NAME
END
```

On an EBCDIC-based platform, this example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME
---------------  ----------
Banning          JOHN
BLACKWOOD        ROSEMARIE
CROSS            BARBARA
Mcknight         ROGER
MCCOY            JOHN
Romans           ANTHONY
```

On an ASCII-based platform, this example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME
---------------  ----------
BLACKWOOD        ROSEMARIE
Banning          JOHN
CROSS            BARBARA
MCCOY            JOHN
Mcknight         ROGER
Romans           ANTHONY
```

In the EBCDIC example, Mcknight appears before MCCOY, since the EBCDIC sorting order places lowercase letters before uppercase letters. In the ASCII example, Blackwood appears before Banning, since the ASCII sorting order places uppercase letters before lowercase letters. In either case, this is not how you would expect your report to be sorted.

The solution is to create a new field with all uppercase letters and sort using this field:

```
DEFINE FILE HOLD
LAST_NAME_UPPER/A15 = UPCASE(15, LAST_NAME, 'A15') ;
END
```

```
TABLE FILE HOLD
PRINT LAST_NAME AND FIRST_NAME BY LAST_NAME_UPPER
END
```

Now, when you execute the report request, the names are sorted correctly:

```
PAGE     1


LAST_NAME_UPPER  LAST_NAME        FIRST_NAME
---------------  ---------------  ----------
BANNING          Banning          JOHN
BLACKWOOD        BLACKWOOD        ROSEMARIE
CROSS            CROSS            BARBARA
MCCOY            MCCOY            JOHN
MCKNIGHT         Mcknight         ROGER
ROMANS           Romans           ANTHONY
```

If you do not want to see the field with all uppercase values, you can NOPRINT it.

Example MODIFY Request Using UPCASE

Suppose your company decided to store employee names in mixed case and the department assignments in uppercase in the EMPLOYEE data source.

To enter records of new employees, execute this MODIFY procedure:

```
MODIFY FILE EMPLOYEE
CRTFORM LOWER
 "ENTER EMPLOYEE'S ID : <EMP_ID"
 "ENTER LAST_NAME: <LAST_NAME FIRST_NAME: <FIRST_NAME"
 "TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET"
 " "
 "ENTER DEPARTMENT ASSIGNMENT: <DEPARTMENT"
MATCH EMP_ID
     ON MATCH REJECT
     ON NOMATCH COMPUTE
        DEPARTMENT = UPCASE (10, DEPARTMENT, 'A10');
     ON NOMATCH INCLUDE
     ON NOMATCH TYPE "DEPARTMENT VALUE CHANGED TO UPPERCASE: <DEPARTMENT"
DATA
END
```

A sample execution is as follows:

```
ENTER EMPLOYEE'S ID :  444555666
ENTER LAST_NAME:  Cutter          FIRST_NAME:  Alan
TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET

ENTER DEPARTMENT ASSIGNMENT:  sales
```

The procedure processes as:

1. The procedure prompts you for an employee ID, last name, first name, and department on a CRTFORM screen. The CRTFORM LOWER option retains the case of entries as they were typed.
2. You type the following data and press the ENTER key:

```
EMPLOYEE'S ID:            444555666
LAST_NAME:                Cutter
FIRST_NAME:               Alan
DEPARTMENT ASSIGNMENT:    sales
```

3. The procedure searches the data source for the ID 444555666. If it does not find the ID, it continues processing the transaction.
4. The UPCASE subroutine converts the DEPARTMENT entry “sales” to “SALES.”

```
ENTER EMPLOYEE'S ID :
ENTER LAST_NAME:                   FIRST_NAME:
TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET

ENTER DEPARTMENT ASSIGNMENT:

DEPARTMENT VALUE CHANGED TO UPPERCASE: SALES
```

5. The procedure adds the transaction to the data source.
6. When you exit the procedure with PF3, the FOCUS transaction message indicates the number of transactions accepted or rejected.

```
TRANSACTIONS:         TOTAL =     1  ACCEPTED=     1  REJECTED=     0

SEGMENTS:             INPUT =     1  UPDATED =     0  DELETED =     0
```

YM: Calculating Elapsed Months

The YM subroutine calculates the number of months that elapse between two dates. The dates must be in year-month format. You can convert a date to this format by using the CHGDAT subroutine or the EDIT function.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

Available on: All platforms.

Related functions and subroutines:

- CHGDAT
- DMY, MDY, YMD

Syntax How to Calculate Elapsed Months

The syntax is

```
YM(fromdate, todate, outfield)
```

where:

fromdate

Numeric

Is the starting date in year-month format (for example, I4YM). If the date is not valid, the subroutine returns a 0.

todate

Numeric

Is the ending date in year-month format. If the date is not valid, the subroutine returns a 0.

outfield

Integer

Is the name of the field to which the number of months between the two dates is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

Tip: If the input date is in integer year-month-day format (I6YMD or I8YYMD), simply divide the date by 100 to convert to year-month format and set the result to be an integer. This causes the day portion of the date, which is now after the decimal point, to be dropped.

Example Report Request Calculating Difference in Months Between Two Dates

The following example shows the number of months that elapse between the time employees get raises and the time they were hired. Note that the COMPUTE expression converts the dates from year-month-day to year-month format by dividing the dates by 100.

```
TABLE FILE EMPLOYEE
PRINT DAT_INC AS 'RAISE DATE' AND COMPUTE
HIRE_MONTH/I4YM = HIRE_DATE/100; NOPRINT AND COMPUTE
MONTH_INC/I4YM = DAT_INC/100; NOPRINT AND COMPUTE
MONTHS_HIRED/I3 = YM(HIRE_MONTH, MONTH_INC, 'I3');
BY LAST_NAME BY FIRST_NAME BY HIRE_DATE
IF MONTHS_HIRED NE 0
END
```

The example produces the following report:

```
PAGE     1


LAST_NAME        FIRST_NAME  HIRE_DATE  RAISE DATE  MONTHS_HIRED
---------        ----------  ---------  ----------  ------------
CROSS            BARBARA      81/11/02    82/04/09             5
GREENSPAN        MARY         82/04/01    82/06/11             2
IRVING           JOAN         82/01/04    82/05/14             4
JONES            DIANE        82/05/01    82/06/01             1
MCCOY            JOHN         81/07/01    82/01/01             6
MCKNIGHT         ROGER        82/02/02    82/05/14             3
SMITH            MARY         81/07/01    82/01/01             6
                 RICHARD      82/01/04    82/05/14             4
STEVENS          ALFRED       80/06/02    82/01/01            19
                                          81/01/01             7
```

10 Customizing Tabular Reports

Topics:

- Creating Paging and Numbering
- Separating Sections of a Report: SKIP-LINE and UNDER-LINE
- Suppressing Fields: SUP-PRINT or NOPRINT
- Creating New Column Titles: AS
- Customizing Column Names: SET QUALTITLES
- Positioning Columns: IN
- Reducing a Report's Width: FOLD-LINE and OVER
- Controlling Column Spacing: SET SPACES
- Column Title Justification
- Customizing Reports With SET Parameters
- Producing Headings and Footings
- Conditionally Formatting Reports With the WHEN Clause
- Controlling the Display of Empty Reports

FOCUS provides a variety of formatting options that enable you to customize your reports. For example, you can specify page breaks, rename report column titles, and add subfoot text to the bottom of pages.

Note: FOCUS formats reports automatically using defaults based on the formats of fields. However, you can override these defaults to customize your report format to suit your individual requirements.

Creating Paging and Numbering

The appearance of your report can be enhanced by controlling paging and page numbering. You can:

- Specify a page break (PAGE-BREAK).
- Reposition page numbers (TABPAGENO).
- Suppress page numbers (SET PAGE).
- Prevent widow lines (NOSPLIT).

Specifying a Page Break: PAGE-BREAK

Use the PAGE-BREAK option to start a new page each time the specified sort field value changes or to prevent information that should be grouped together from being presented over more than one page.

To specify a page break, use PAGE-BREAK in either an ON phrase or BY phrase immediately after the sort field on which you want to break the page. You can also use PAGE-BREAK to:

- Reset the report page to 1 at specified points (REPAGE).
- Specify conditional page breaks in the printing of a report (with WHEN).

Syntax How to Specify a Page Break

The syntax is

```
{ON|BY} fieldname PAGE-BREAK [REPAGE][WHEN expression;]
```

where:

fieldname
Is a sort field. A change in the sort field value causes a page-break.

REPAGE
Resets the page number to 1 at the sort break or, if WHEN is used, whenever the conditions in the WHEN clause are met.

WHEN *expression*
Specifies conditional page breaks in the printing of a report as determined by a Boolean expression (see *Using Direct Operators in Headings and Footings* on page 10-45).

Reference Usage Notes for Page Breaks

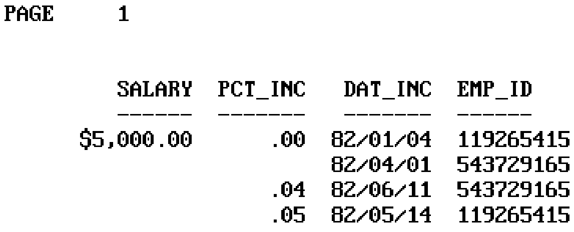
- Page headings and column titles will appear at the top of each new page.
- Put the PAGE-BREAK command on the lowest-level sort field at which the page break is to occur.
- Page breaks automatically occur whenever a higher-level sort field changes.
- PAGE-BREAK is ignored when report output is stored in HOLD, SAVE, or SAVB files (see Chapter 12, *Saving and Reusing Your Report Output*).

Example Specifying a Page Break

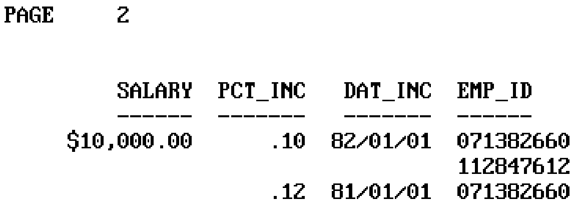
For example:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID
BY SALARY IN-GROUPS-OF 5000
BY PCT_INC BY DAT_INC
ON SALARY PAGE-BREAK
END
```

The first two pages of this report display as:



```
PAGE     1


         SALARY  PCT_INC   DAT_INC  EMP_ID
         ------  -------   -------  ------
      $5,000.00      .00  82/01/04  119265415
                          82/04/01  543729165
                     .04  82/06/11  543729165
                     .05  82/05/14  119265415
```



```
PAGE     2


         SALARY  PCT_INC   DAT_INC  EMP_ID
         ------  -------   -------  ------
     $10,000.00      .10  82/01/01  071382660
                                    112847612
                     .12  81/01/01  071382660
```

Inserting Page Numbers: TABPAGENO

By default, FOCUS reserves the first two lines of each report page: the first line contains the page number at the left margin—that is, in the top-left corner of the page—and the following line is blank. You can change the position of the page number with the TABPAGENO system variable.

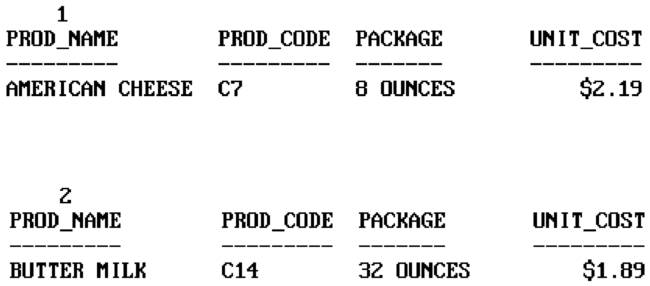
TABPAGENO contains the page number of the current page and acts like a field name. Therefore, it can be positioned in a heading or footing (or subhead/subfoot). The default page number in the top left-hand corner is automatically suppressed when this variable is used.

Example Inserting Page Numbers

For example, this request

```
TABLE FILE PROD
"<TABPAGENO"
PRINT PACKAGE AND UNIT_COST
BY PROD_NAME BY PROD_CODE
ON PROD_NAME PAGE-BREAK
END
```

creates the following report (of which the first two pages are shown):



```
    1
PROD_NAME        PROD_CODE  PACKAGE        UNIT_COST
---------        ---------  -------        ---------
AMERICAN CHEESE  C7         8 OUNCES           $2.19



    2
PROD_NAME        PROD_CODE  PACKAGE        UNIT_COST
---------        ---------  -------        ---------
BUTTER MILK      C14        32 OUNCES          $1.89
```

Note that FOCUS continues to reserve the top two lines of every report page.

Suppressing Page Numbers: SET PAGE

Automatic page numbering can also be suppressed with the SET PAGE command:

Syntax How to Suppress Page Numbers

To suppress page numbering, the syntax is

```
SET PAGE = {OFF|NOPAGE}
```

where:

`OFF`

Suppresses automatic page numbering. You can still use the variable TABPAGENO as described in *Inserting Page Numbers: TABPAGENO* on page 10-4. Note that FOCUS reserves the top two lines of every page.

`NOPAGE`

Suppresses all page indicators and makes the first two lines of each report page available for your use. NOPAGE does not issue page ejects; they are issued if you use SET PAGE=OFF.

Preventing Widow Lines: NOSPLIT

Page breaks sometimes occur where report information has been logically grouped by sort field(s), causing one or two lines to appear by themselves on the next page or screen. To prevent these widow lines, use NOSPLIT in either an ON phrase or immediately after the first reference to the sort field in a BY phrase.

Syntax How to Prevent Widow Lines

The syntax is

```
{ON|BY} fieldname NOSPLIT
```

where:

`fieldname`

Is the name of the sort field for which sort groups will be kept together on the same page.

Whenever the value of the specified field changes, FOCUS determines if the total number of lines related to the new value can fit on the current page. If they cannot, the page breaks and the group of lines appears on the next page.

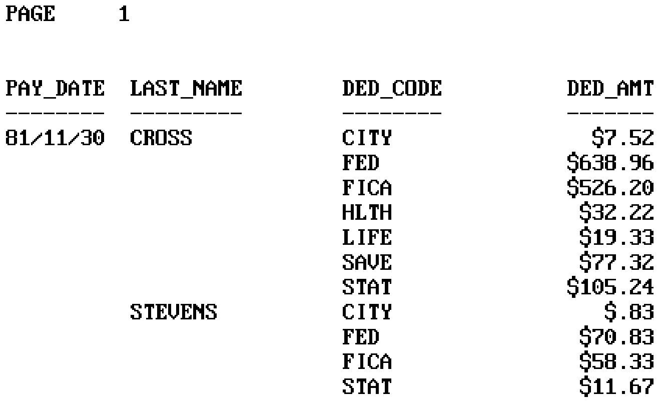
Reference Usage Notes for Preventing Widow Lines

- Only one NOSPLIT option is allowed per report. If a PAGE-BREAK option also exists in the request, it must relate to a higher-level sort field; otherwise, NOSPLIT will be ignored.
- Subtotals, footings, subheads, and subfoots are placed on the same page as the detail lines; headings are placed on the new page.
- NOSPLIT is ignored when report output is stored in HOLD, SAVE, or SAVB files (see Chapter 12, *Saving and Reusing Your Report Output*).
- NOSPLIT is not compatible with the TABLEF command and produces an FOC037 error message.

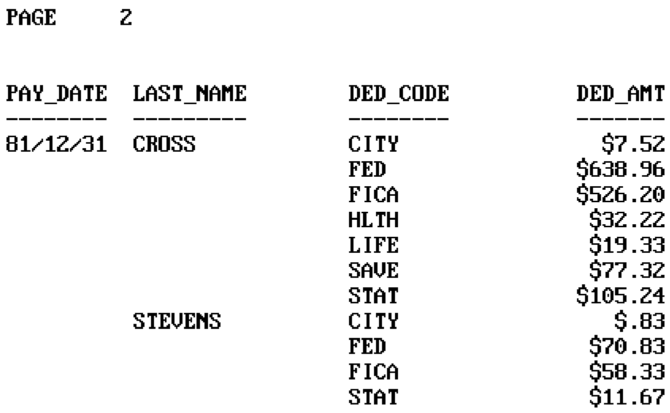
Example Preventing Widow Lines

```
TABLE FILE EMPLOYEE
PRINT DED_CODE AND DED_AMT
BY PAY_DATE BY LAST_NAME
ON LAST_NAME NOSPLIT
END
```

Depending upon how many lines your output device is set to, the first two pages of the previous request might display as:

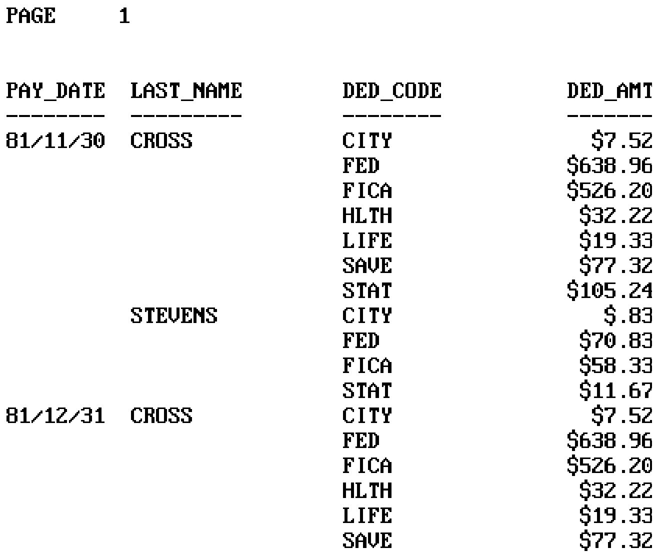


```
PAGE     1


PAY_DATE  LAST_NAME         DED_CODE          DED_AMT
--------  ---------         --------          -------
81/11/30  CROSS             CITY                $7.52
                            FED               $638.96
                            FICA              $526.20
                            HLTH               $32.22
                            LIFE               $19.33
                            SAVE               $77.32
                            STAT              $105.24
          STEVENS           CITY                 $.83
                            FED                $70.83
                            FICA               $58.33
                            STAT               $11.67
```

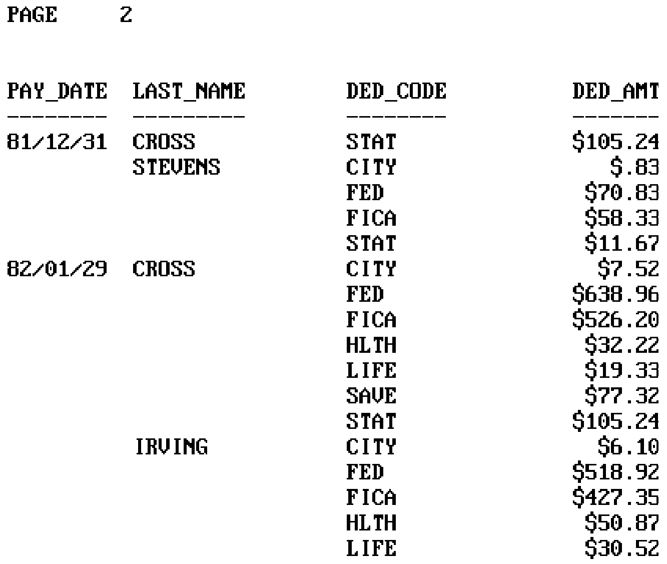


```
PAGE     2

PAY_DATE  LAST_NAME         DED_CODE          DED_AMT
--------  ---------         --------          -------
81/12/31  CROSS             CITY                $7.52
                            FED               $638.96
                            FICA              $526.20
                            HLTH               $32.22
                            LIFE               $19.33
                            SAVE               $77.32
                            STAT              $105.24
          STEVENS           CITY                 $.83
                            FED                $70.83
                            FICA               $58.33
                            STAT               $11.67
```

Here are the first two pages without NOSPLIT:



```
PAGE     1

PAY_DATE  LAST_NAME         DED_CODE          DED_AMT
--------  ---------         --------          -------
81/11/30  CROSS             CITY                $7.52
                            FED               $638.96
                            FICA              $526.20
                            HLTH               $32.22
                            LIFE               $19.33
                            SAVE               $77.32
                            STAT              $105.24
          STEVENS           CITY                 $.83
                            FED                $70.83
                            FICA               $58.33
                            STAT               $11.67
81/12/31  CROSS             CITY                $7.52
                            FED               $638.96
                            FICA              $526.20
                            HLTH               $32.22
                            LIFE               $19.33
                            SAVE               $77.32
```



```
PAGE     2


PAY_DATE  LAST_NAME        DED_CODE          DED_AMT
--------  ---------        --------          -------
81/12/31  CROSS            STAT              $105.24
          STEVENS          CITY                 $.83
                           FED                $70.83
                           FICA               $58.33
                           STAT               $11.67
82/01/29  CROSS            CITY                $7.52
                           FED               $638.96
                           FICA              $526.20
                           HLTH               $32.22
                           LIFE               $19.33
                           SAVE               $77.32
                           STAT              $105.24
          IRVING           CITY                $6.10
                           FED               $518.92
                           FICA              $427.35
                           HLTH               $50.87
                           LIFE               $30.52
```

The report without NOSPLIT has a widow line for Cross on Page 2, whereas the report using NOSPLIT does not.

Separating Sections of a Report: SKIP-LINE and UNDER-LINE

To make a detailed report easier to read and interpret, you can separate sections of it—individual lines, or entire sort groups—by inserting blank lines between them, or (for sort groups only) by underlining them.

Adding Blank Lines: SKIP-LINE

Report information often stands out more clearly if lines are skipped between individual lines, or between sort groups. You can use SKIP-LINE with either a sort field or a display field.

- If you use SKIP-LINE with a sort field, FOCUS inserts a blank line between each section of the report.
- If you use SKIP-LINE with a display field, FOCUS inserts a blank line between each line of the report—in effect, double-spacing the report. Double spacing is especially helpful when a report will be used as a review document, as it makes it easy for the reader to write comments next to individual lines.

Syntax How to Add Blank Lines

To add blank lines, use SKIP-LINE with the keyword ON, or BY. Use the WHEN clause to specify conditional blank lines in the printing of a report. The syntax is

```
display fieldname SKIP-LINE
{ON|BY} fieldname SKIP-LINE [WHEN expression;]
```

where:

display

Is any display command.

fieldname

Is used so that when the value of this field changes, a blank line is inserted before the next set of values.

WHEN *expression*

Specifies conditional blank lines in the printing of a report as determined by a Boolean expression (see *Using Direct Operators in Headings and Footings* on page 10-45).

You can use only one SKIP-LINE in each report request. You do not have to enter it on its own line; instead, include it after the field name or sort field for which you want to insert a blank line.

Reference Usage Notes for Adding Blank Lines

Keep the following in mind when using SKIP-LINE:

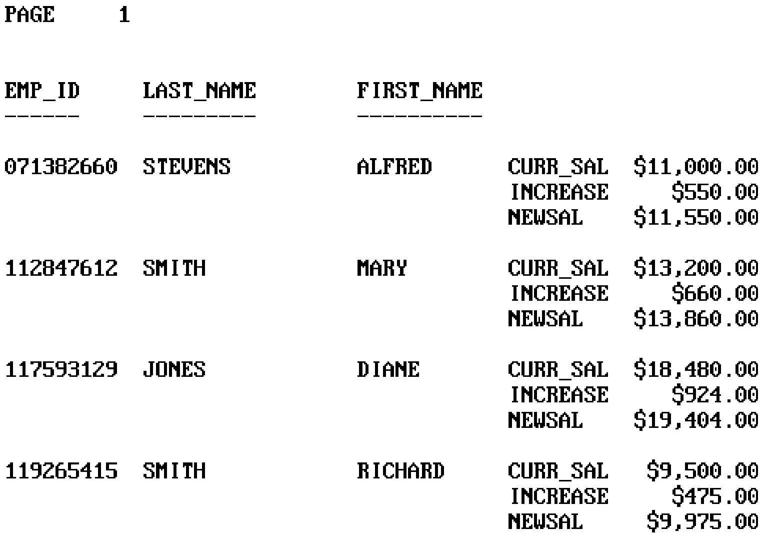
- If the field name is a sort field, a blank line is inserted just before every change in value of the sort field.
- If the field name is a display field, a blank line is inserted after every printed line. The WHEN clause does not apply to display fields.
- This is one of the only ON conditions that does not have to refer solely to sort control (BY) fields.
- Only one SKIP-LINE option is allowed per request and it may affect more than one sort field.

Example Adding Blank Lines

For example:

```
DEFINE FILE EMPLOYEE
INCREASE/D8.2M = .05*CURR_SAL;
CURR_SAL/D8.2M=CURR_SAL;
NEWSAL/D8.2M=CURR_SAL + INCREASE;
END

TABLE FILE EMPLOYEE
PRINT CURR_SAL OVER INCREASE OVER NEWSAL
BY EMP_ID BY LAST_NAME BY FIRST_NAME
ON EMP_ID SKIP-LINE
END
```

The first part of the report output is shown below:



```
PAGE     1


EMP_ID     LAST_NAME        FIRST_NAME
------     ---------        ----------

071382660  STEVENS          ALFRED      CURR_SAL  $11,000.00
                                        INCREASE     $550.00
                                        NEWSAL    $11,550.00

112847612  SMITH            MARY        CURR_SAL  $13,200.00
                                        INCREASE     $660.00
                                        NEWSAL    $13,860.00

117593129  JONES            DIANE       CURR_SAL  $18,480.00
                                        INCREASE     $924.00
                                        NEWSAL    $19,404.00

119265415  SMITH            RICHARD     CURR_SAL   $9,500.00
                                        INCREASE     $475.00
                                        NEWSAL     $9,975.00
```

Underlining Values: UNDER-LINE

Drawing a line across the page after all of the information for a particular section has been displayed can enhance the readability of a printed report.

Syntax

How to Underline Values

The syntax is

```
{ON|BY} fieldname UNDER-LINE [WHEN expression;]
```

where:

fieldname

Is used so that when the value of the field changes, a line is drawn. A line is automatically drawn after any other option such as RECAP or SUB-TOTAL (but before PAGE-BREAK).

WHEN *expression*

Specifies conditional underlines in the printing of a report, as determined by a Boolean expression (see *Inserting Page Numbers: TABPAGENO* on page 10-4).

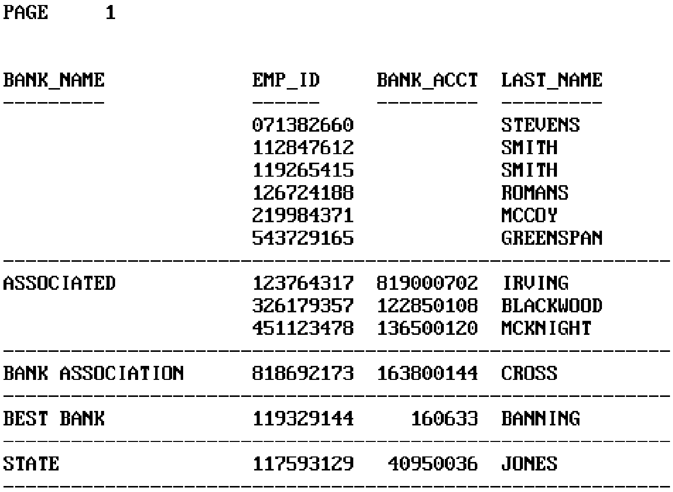
Example

Underlining Values

For example:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND BANK_ACCT AND LAST_NAME
BY BANK_NAME
ON BANK_NAME UNDER-LINE
END
```

The request produces the following report:



```
PAGE     1


BANK_NAME             EMP_ID     BANK_ACCT  LAST_NAME
---------             ------     ---------  ---------
                      071382660             STEVENS
                      112847612             SMITH
                      119265415             SMITH
                      126724188             ROMANS
                      219984371             MCCOY
                      543729165             GREENSPAN
-----------------------------------------------------------
ASSOCIATED            123764317  819000702  IRVING
                      326179357  122850108  BLACKWOOD
                      451123478  136500120  MCKNIGHT
-----------------------------------------------------------
BANK ASSOCIATION      818692173  163800144  CROSS
-----------------------------------------------------------
BEST BANK             119329144     160633  BANNING
-----------------------------------------------------------
STATE                 117593129   40950036  JONES
-----------------------------------------------------------
```

Suppressing Fields: SUP-PRINT or NOPRINT

You can create reports that do not display the values or titles of fields, but only use those fields to produce specific effects. FOCUS provides options to suppress the printing of field values: NOPRINT and SUP-PRINT.

Syntax How to Suppress Fields

The syntax is:

```
display fieldname {SUP-PRINT|NOPRINT}
{ON|BY} fieldname {SUP-PRINT|NOPRINT}
```

Valid values are:

display

Is any display command.

fieldname

Is a sort field or display field. The values of the field may be used, but they will not be displayed.

Reference Usage Notes for Suppressing Fields

- If you put a NOPRINT or SUP-PRINT phrase in a computed field, you must then repeat AND COMPUTE before the next computed field.
- If you use the NOPRINT option with a BY field and create a HOLD file, the BY field is excluded from the file. For example, a request that includes the phrase

  ```
  BY DEPARTMENT NOPRINT
  ```

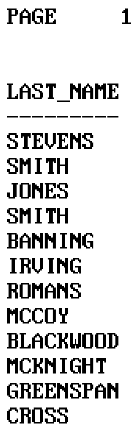
  will result in a HOLD file that does not contain the DEPARTMENT field.

Example Suppressing Fields

For example, to print a list of employee names in alphabetical order, if you simply used the request

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

you would get a report that lists the last names of employees in the order they were entered into the data source:

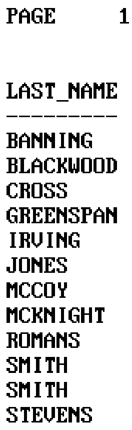


```
PAGE     1


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

To print the last names in alphabetical order, use NOPRINT in conjunction with a BY phrase

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

which produces the desired result:



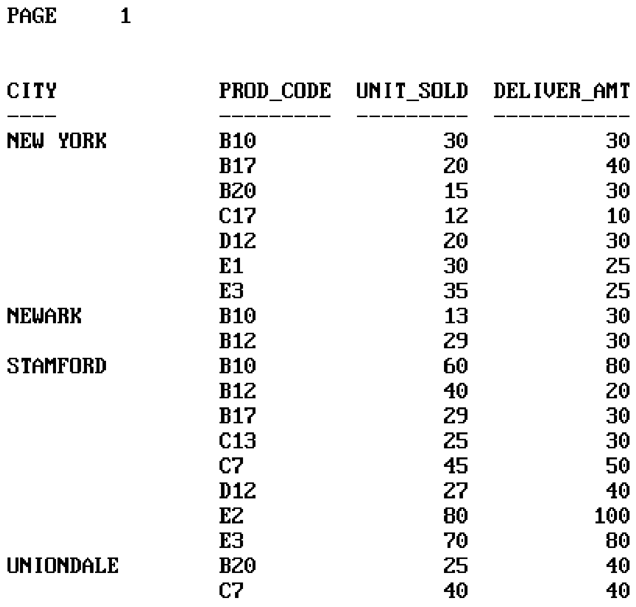
```
PAGE     1


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

Example Suppressing a Sort Field

Consider the following example, where the report is sorted, but the field that determines the sort order is not displayed:

```
TABLE FILE SALES
PRINT UNIT_SOLD AND DELIVER_AMT
BY CITY BY PROD_CODE BY RETAIL_PRICE
ON RETAIL_PRICE SUP-PRINT
END
```

The report is as follows:

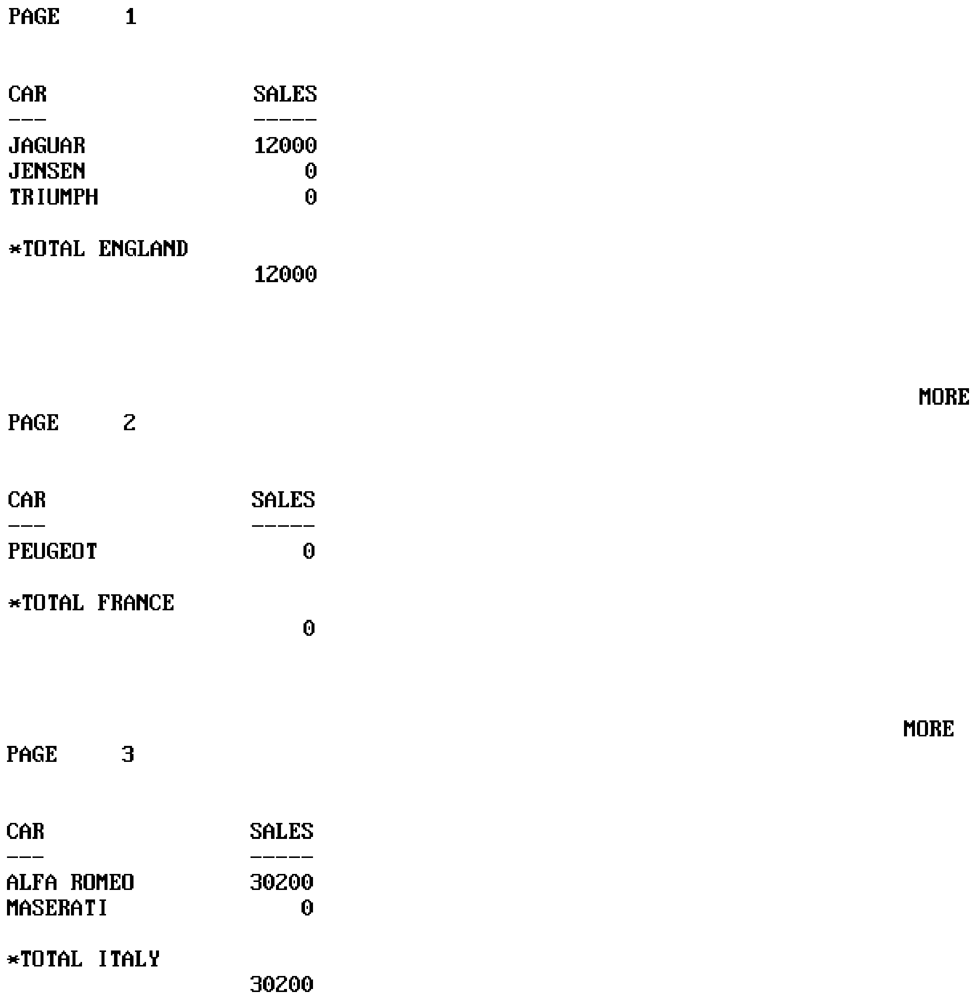


```
PAGE     1


CITY             PROD_CODE  UNIT_SOLD  DELIVER_AMT
----             ---------  ---------  -----------
NEW YORK         B10               30           30
                 B17               20           40
                 B20               15           30
                 C17               12           10
                 D12               20           30
                 E1                30           25
                 E3                35           25
NEWARK           B10               13           30
                 B12               29           30
STAMFORD         B10               60           80
                 B12               40           20
                 B17               29           30
                 C13               25           30
                 C7                45           50
                 D12               27           40
                 E2                80          100
                 E3                70           80
UNIONDALE        B20               25           40
                 C7                40           40
```

Also, consider the following example

```
TABLE FILE CAR
SUM SALES BY COUNTRY
BY CAR
ON COUNTRY SUB-TOTAL SUP-PRINT PAGE-BREAK
END
```

which generates this report (only the beginning is shown):



```
PAGE     1


CAR                SALES
---                -----
JAGUAR             12000
JENSEN                 0
TRIUMPH                0

*TOTAL ENGLAND
                   12000



                                                                       MORE
PAGE     2


CAR                SALES
---                -----
PEUGEOT                0

*TOTAL FRANCE
                       0



                                                                       MORE
PAGE     3


CAR                SALES
---                -----
ALFA ROMEO         30200
MASERATI               0

*TOTAL ITALY
                   30200
```

Creating New Column Titles: AS

Use the AS option to rename existing column titles in your reports. Any of the following titles can be changed with an AS phrase:

- ACROSS titles can be replaced by one line of text only.
- A SUBTOTAL line can be replaced by one line of text only.
- FOR phrases.
- Fields for the MATCH command.

Syntax **How to Create Column Titles**

The syntax for changing default titles is

```
field AS 'title1, title2,...'
```

where:

field
Can be a sort field, display field, column total, or row total.

title
Is the new column title enclosed in single quotation marks.

To specify multiple lines in a column title, separate each line’s text with commas. Up to five lines are allowed.

Reference **Usage Notes for New Column Titles**

- When using FOLD-LINE, the titles appear one over the other. No more than one line per title is allowed with FOLD-LINE. (See *Reducing a Report’s Width: FOLD-LINE and OVER* on page 10-23.)
- The use of a title line larger than the format size of the data is one convenient way to space out a report across the columns of the page. For instance,

  ```
  PRINT UNITS BY MONTH AS ' MONTH'
  ```

  shifts the title for MONTH to the right and all other columns, in this case UNITS, shift to the right. For more information on changing the column position, see *Reducing a Report’s Width: FOLD-LINE and OVER* on page 10-23.
- If you do not want any field name or title displayed in the report, you can also use the AS phrase by entering two consecutive single quotation marks. For example:

  ```
  PRINT LAST_NAME AS ''
  ```

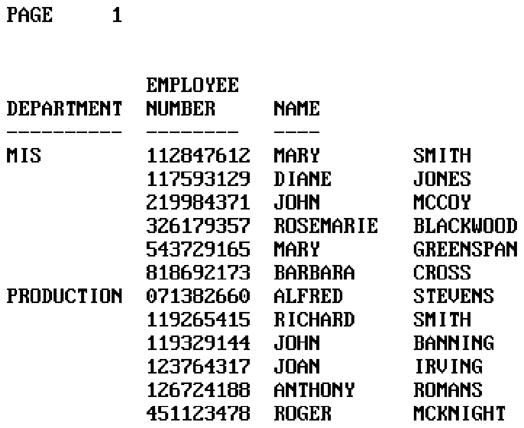
  To display underscores, enclose blanks in single quotation marks.
- If you put an AS phrase in a computed field, you must then repeat the keyword COMPUTE before the next computed field.

Example Creating New Column Titles

For example:

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AS 'NAME' AND LAST_NAME AS ''
BY DEPARTMENT
BY EMP_ID AS 'EMPLOYEE,NUMBER'
END
```

This request produces the following report:



```
PAGE     1


            EMPLOYEE
DEPARTMENT  NUMBER     NAME
----------  --------   ----
MIS         112847612  MARY        SMITH
            117593129  DIANE       JONES
            219984371  JOHN        MCCOY
            326179357  ROSEMARIE   BLACKWOOD
            543729165  MARY        GREENSPAN
            818692173  BARBARA     CROSS
PRODUCTION  071382660  ALFRED      STEVENS
            119265415  RICHARD     SMITH
            119329144  JOHN        BANNING
            123764317  JOAN        IRVING
            126724188  ANTHONY     ROMANS
            451123478  ROGER       MCKNIGHT
```

Customizing Column Names: SET QUALTITLES

The FOCUS SET command, SET QUALTITLES, enables you to determine whether or not duplicate field names appear as qualified column titles in TABLE output. (For more information about long and qualified field names, see Chapter 2, *Displaying Report Data*.)

Syntax How to Customize Column Headings

The syntax is

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

where:

`ON`

Enables qualified column titles when duplicate field names exist and FIELDNAME is set to NEW.

`OFF`

Disables qualified column titles. OFF is the default value.

SET QUALTITLES may also be issued from within a TABLE request.

Qualified column titles are automatically used, even if qualified field names are not used in the request.

Reference Usage Notes for Qualified Column Titles

- AS names are used if duplicate field names are referenced in a MATCH request.
- AS names are used when duplicate field names exist in a HOLD file.

Positioning Columns: IN

FOCUS automatically formats a page and uses common default values for determining column positions and spacing. You can override these defaults by specifying the absolute or relative column position where a data value is to appear on a report.

Syntax How to Position Columns

The syntax is

```
field IN  {n|+n}
```

Valid values are:

field
Is the field (that is, the column) that you want to move.

n
Is a number indicating the absolute position of the column.

+n
Is a number indicating the relative position of the column. That is, +*n* is the number of characters to the right of the last column.

Reference Usage Notes for Positioning Columns

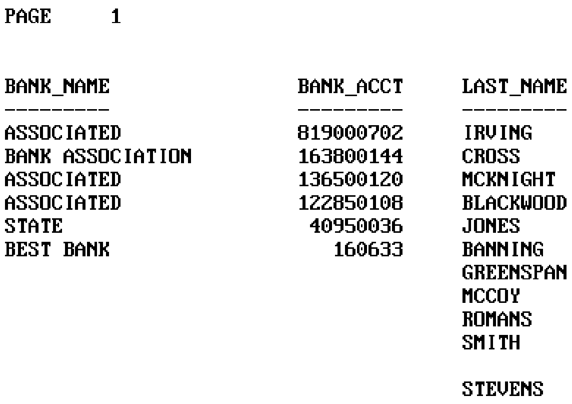
- The IN phrase can be used with ACROSS to specify both the starting column of the entire ACROSS set as well as the spacing between each column within the ACROSS.
- When one field is positioned over another (for example, when OVER or FOLD-LINE is used; see *Reducing a Report's Width: FOLD-LINE and OVER* on page 10-23), the positions apply to the line on which the referenced field occurs.

Example Positioning Columns

For example:

```
TABLE FILE EMPLOYEE
PRINT BANK_NAME IN 1
BY HIGHEST BANK_ACCT IN 26
BY LAST_NAME IN 40
END
```

This request produces the following report. There is a blank line following SMITH because LAST_NAME is a sort field and there are two employees named Smith in the database.



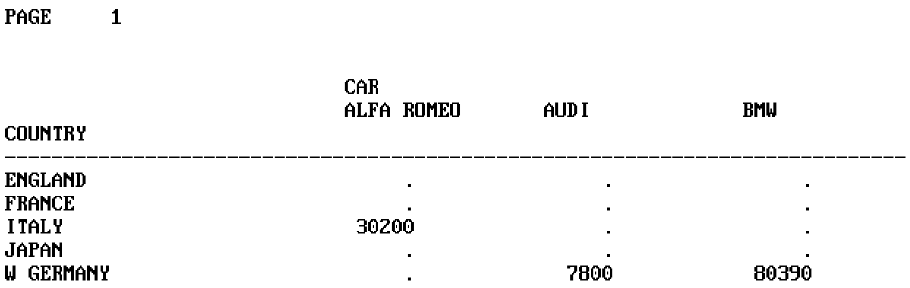
```
PAGE     1


BANK_NAME                BANK_ACCT     LAST_NAME
---------                ---------     ---------
ASSOCIATED               819000702     IRVING
BANK ASSOCIATION         163800144     CROSS
ASSOCIATED               136500120     MCKNIGHT
ASSOCIATED               122850108     BLACKWOOD
STATE                     40950036     JONES
BEST BANK                   160633     BANNING
                                       GREENSPAN
                                       MCCOY
                                       ROMANS
                                       SMITH

                                       STEVENS
```

Example Positioning Columns With ACROSS

The IN phrase can be used with ACROSS to specify both the starting column of the entire ACROSS set as well as the spacing between each column within the ACROSS, as shown in the following example:

```
TABLE FILE CAR
SUM UNITS IN +1 ACROSS CAR IN 30
BY COUNTRY
END
```

This will place one extra space between the data columns in the matrix and display the ACROSS sets beginning in Position 30, as shown in the first page of the report below.



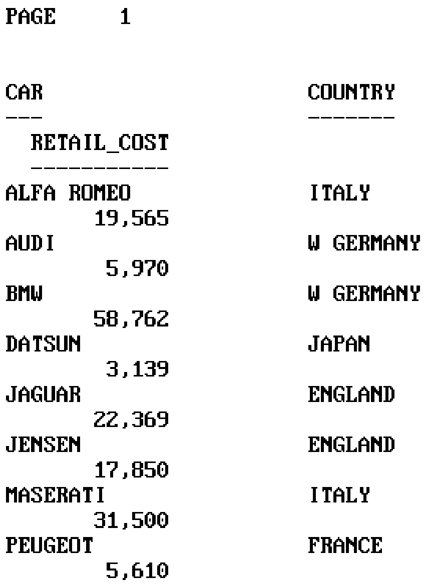
```
PAGE     1


                             CAR
                             ALFA ROMEO       AUDI             BMW
COUNTRY
-----------------------------------------------------------------------------
ENGLAND                           .                .                .
FRANCE                            .                .                .
ITALY                         30200                .                .
JAPAN                             .                .                .
W GERMANY                         .             7800            80390
```

Example Positioning Columns With FOLD-LINE

When one field is positioned over another (for example, when OVER or FOLD-LINE is used, see *Reducing a Report's Width: FOLD-LINE and OVER* on page 10-23), the positions apply to the line on which the referenced field occurs, as in the following example

```
TABLE FILE CAR
SUM RCOST BY CAR
BY COUNTRY IN 25
ON COUNTRY FOLD-LINE
END
```

which creates this report, in which COUNTRY starts in column 25 and RCOST appears on the second line.



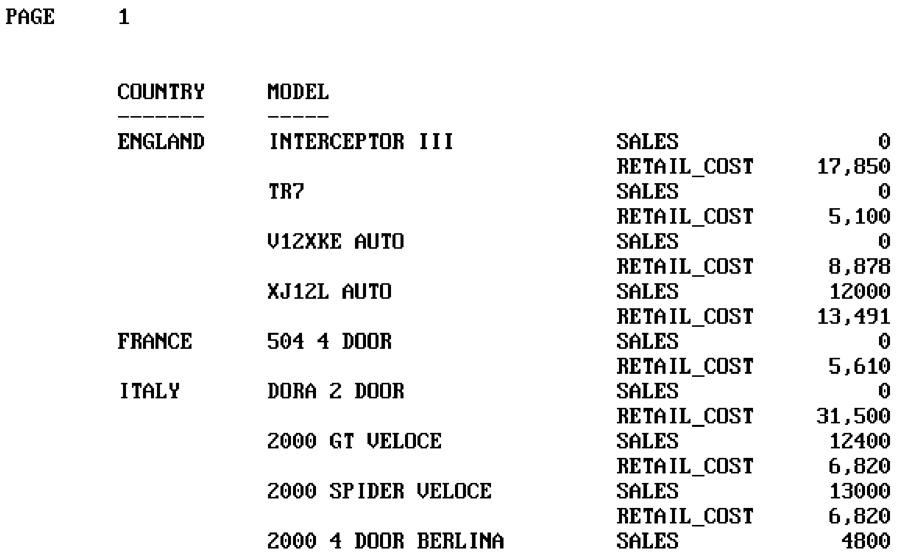
```
PAGE     1


CAR                     COUNTRY
---                     -------
  RETAIL_COST
  -----------
ALFA ROMEO              ITALY
       19,565
AUDI                    W GERMANY
        5,970
BMW                     W GERMANY
       58,762
DATSUN                  JAPAN
        3,139
JAGUAR                  ENGLAND
       22,369
JENSEN                  ENGLAND
       17,850
MASERATI                ITALY
       31,500
PEUGEOT                 FRANCE
        5,610
```

Example Positioning Columns With OVER

Consider the following report request

```
TABLE FILE CAR
PRINT SALES IN 50 OVER RCOST IN 50
BY COUNTRY IN 10 BY MODEL
END
```

which generates this report.



```
PAGE     1

         COUNTRY    MODEL
         -------    -----
         ENGLAND    INTERCEPTOR III             SALES                0
                                                RETAIL_COST     17,850
                    TR7                         SALES                0
                                                RETAIL_COST      5,100
                    V12XKE AUTO                 SALES                0
                                                RETAIL_COST      8,878
                    XJ12L AUTO                  SALES            12000
                                                RETAIL_COST     13,491
         FRANCE     504 4 DOOR                  SALES                0
                                                RETAIL_COST      5,610
         ITALY      DORA 2 DOOR                 SALES                0
                                                RETAIL_COST     31,500
                    2000 GT VELOCE              SALES            12400
                                                RETAIL_COST      6,820
                    2000 SPIDER VELOCE          SALES            13000
                                                RETAIL_COST      6,820
                    2000 4 DOOR BERLINA         SALES             4800
```

Reducing a Report’s Width: FOLD-LINE and OVER

Wide reports are difficult to read, especially on a screen. To reduce a report’s width, use FOLD-LINE and OVER.

Compressing the Columns of Reports: FOLD-LINE

A single line on a report can be folded to compress it into fewer columns. This enables you to display a wide report on a narrow screen and enhance the appearance of many reports which might otherwise have wasted space under sort control fields which change infrequently.

Syntax How to Compress Report Columns

The syntax for specifying the point of line fold is

```
display fieldname ... FOLD-LINE fieldname ...
```

```
{ON|BY} fieldname FOLD-LINE
```

where:

`display`

Is any display command.

`fieldname`

Causes columns to be placed on a separate line when the value of the field changes in the BY or ON phrase. The field name may be a sort field or display field. When it is a display field, it will be placed under the preceding field.

Reference Usage Notes for Compressing Report Columns

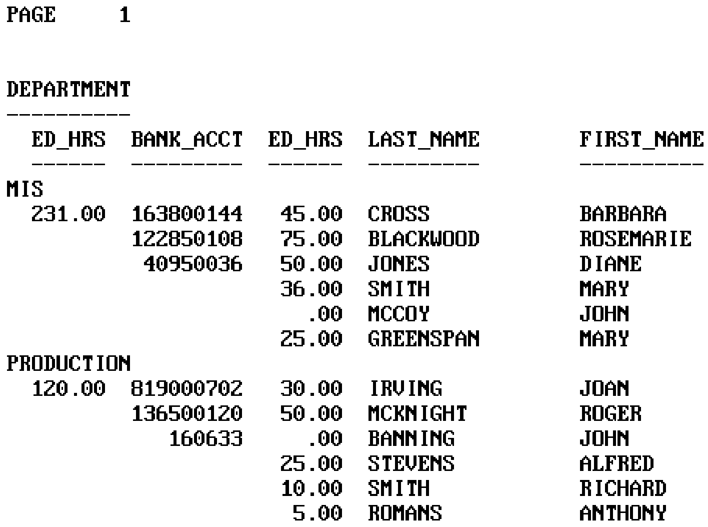
- The second half of the folded line is offset by two spaces from the first part when the line is folded on a sort control field.
- Instead of FOLD-LINE, you can also use the OVER phrase to decrease the width of reports, described in *Decreasing the Width of a Report: OVER* on page 10-24.
- When the point of line folding is after a display field, there is no offset. A simple way to change the line alignment is to use a title with leading blanks. (See *Creating New Column Titles: AS* on page 10-16.)
- The FOLD-LINE option can be specified only once in a report request.

Example Compressing Report Columns

For example:

```
TABLE FILE EMPLOYEE
SUM ED_HRS BY DEPARTMENT
PRINT ED_HRS AND LAST_NAME AND FIRST_NAME
BY DEPARTMENT BY HIGHEST BANK_ACCT
ON DEPARTMENT FOLD-LINE
END
```

This request produces the following report.



```
PAGE     1


DEPARTMENT
----------
  ED_HRS  BANK_ACCT  ED_HRS  LAST_NAME        FIRST_NAME
  ------  ---------  ------  ---------        ----------
MIS
  231.00  163800144   45.00  CROSS            BARBARA
          122850108   75.00  BLACKWOOD        ROSEMARIE
           40950036   50.00  JONES            DIANE
                      36.00  SMITH            MARY
                        .00  MCCOY            JOHN
                      25.00  GREENSPAN        MARY
PRODUCTION
  120.00  819000702   30.00  IRVING           JOAN
          136500120   50.00  MCKNIGHT         ROGER
             160633     .00  BANNING          JOHN
                      25.00  STEVENS          ALFRED
                      10.00  SMITH            RICHARD
                       5.00  ROMANS           ANTHONY
```

Decreasing the Width of a Report: OVER

One way to decrease the width of your report (particularly when using the ACROSS phrase) is to use OVER. OVER places field names over one another. The syntax is

```
display_command fieldname1 OVER fieldname2 OVER fieldname3 ...
```

where:

display
: Is any display command.

fieldname
: Causes the fields listed to be placed over each other, instead of printed beside each other in a row. The field names must be a display field.

Reference Usage Notes for Decreasing Report Width

Keep the following in mind when using OVER:

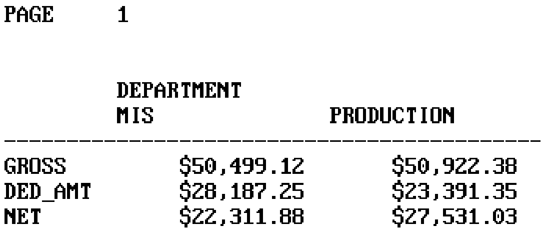
- For more complex combinations of IN and OVER, you may want to create subfoots with data. Subfoots with data are discussed in *Producing Headings and Footings* on page 10-30.
- Text fields cannot be specified with OVER.

Example Decreasing the Width of a Report

For example:

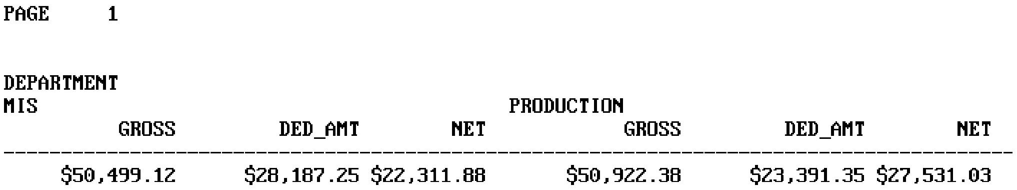
```
TABLE FILE EMPLOYEE
SUM GROSS OVER DED_AMT OVER
COMPUTE NET/D8.2M = GROSS - DED_AMT;
ACROSS DEPARTMENT
END
```

The request produces the following report. Notice the ACROSS values display to the left, not directly above the data values.



```
PAGE     1


         DEPARTMENT
         MIS              PRODUCTION
-------------------------------------------
GROSS         $50,499.12       $50,922.38
DED_AMT       $28,187.25       $23,391.35
NET           $22,311.88       $27,531.03
```

Without the OVER phrase, the report would look like this:



```
PAGE     1


DEPARTMENT
MIS                                         PRODUCTION
          GROSS         DED_AMT        NET            GROSS         DED_AMT        NET
----------------------------------------------------------------------------------------
     $50,499.12      $28,187.25 $22,311.88       $50,922.38      $23,391.35 $27,531.03
```

Controlling Column Spacing: SET SPACES

By default, FOCUS puts one or two spaces between report columns, depending on the output width. The SET SPACES command enables you to control the number of spaces between columns in a report.

Syntax How to Control Column Spacing

The syntax is:

```
SET SPACES = {n|AUTO}
```

Valid values are:

n

Is a number indicating from 1 to 8 spaces.

AUTO

Specifies that FOCUS automatically puts one or two spaces between columns depending on report output and available output length. AUTO is the default setting.

SET SPACES may also be issued from within a TABLE request.

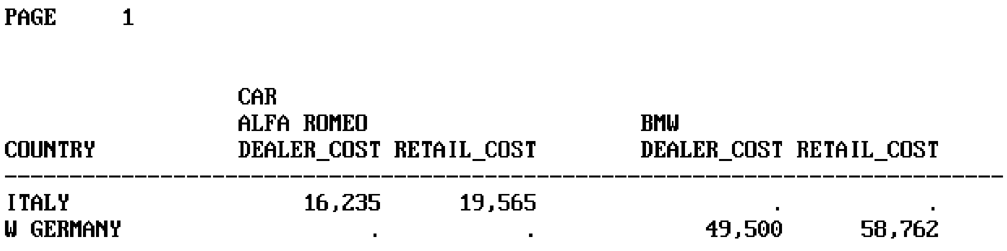
For ACROSS phrases, SET SPACES *n* controls the distance between ACROSS sets. Within an ACROSS set, the distance between fields is always one space and cannot be changed.

Example Controlling Column Spacing

The following example illustrates the use of ACROSS with SET SPACES:

```
TABLE FILE CAR
SUM DEALER_COST RETAIL_COST ACROSS CAR BY COUNTRY
IF CAR EQ 'ALFA ROMEO' OR 'BMW'
ON TABLE SET SPACES 8
END
```

The ACROSS set consists of the fields DEALER_COST and RETAIL_COST. The distance between each set is eight spaces.



```
PAGE     1


                  CAR
                  ALFA ROMEO                          BMW
COUNTRY           DEALER_COST RETAIL_COST        DEALER_COST RETAIL_COST
-----------------------------------------------------------------------------
ITALY                  16,235      19,565                  .           .
W GERMANY                   .           .             49,500      58,762
```

Column Title Justification

You can specify whether column titles in a report are left justified, right justified, or centered. By default, column titles for alphanumeric fields are left justified, and column titles for numeric and date fields are right justified over the displayed column.

Syntax How to Justify Column Titles

The syntax to alter default justification is

```
fieldname [alignment] [/format]
```

where:

alignment

Specifies the alignment of the column title.

`/R` specifies that the column title is to be right justified.

`/L` specifies that the column title is to be left justified.

`/C` specifies that the column title is to be centered.

/format

Is an optional format specification for the field.

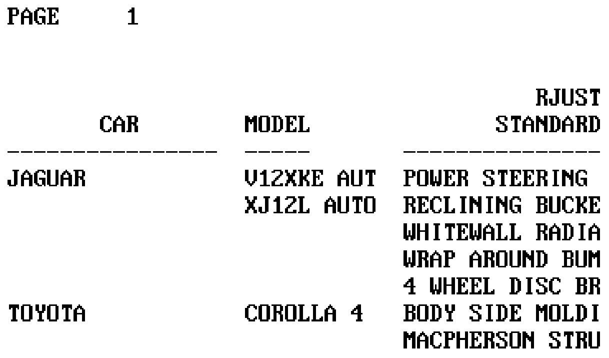
Reference Usage Notes for Justifying Column Titles

- You may specify justification for display fields, BY fields, ACROSS fields, column totals, and row totals (see Chapter 2, *Displaying Report Data*). For ACROSS fields, data values, not column titles, are justified as specified.
- For display commands and row totals only, the justification parameter may be combined with a format specification, which precedes or follows the justification parameter (for example, PRINT CAR/A8/R MODEL/C/A15).
- If a title is specified with an AS phrase or in the Master File, that title will be justified as specified for the field.
- When multiple ACROSS fields are requested, justification is performed on the lowest ACROSS level only. All other justification parameters for ACROSS fields are ignored.

Example Justifying Column Titles

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

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



```
PAGE     1


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

Customizing Reports With SET Parameters

Most SET commands that change system defaults may be issued from within a report request. Many SET command parameters can be used to enhance the readability and usefulness of your reports. The SET command, when used in this manner, affects only the request in which it occurs. For a complete list of SET parameters and acceptable values, see the *Developing Applications* manual.

Syntax How to Use SET Parameters in Requests

The syntax is

```
ON TABLE SET parameter value [AND parameter value...]
```

where:

parameter

Is the SET command parameter that you wish to change.

value

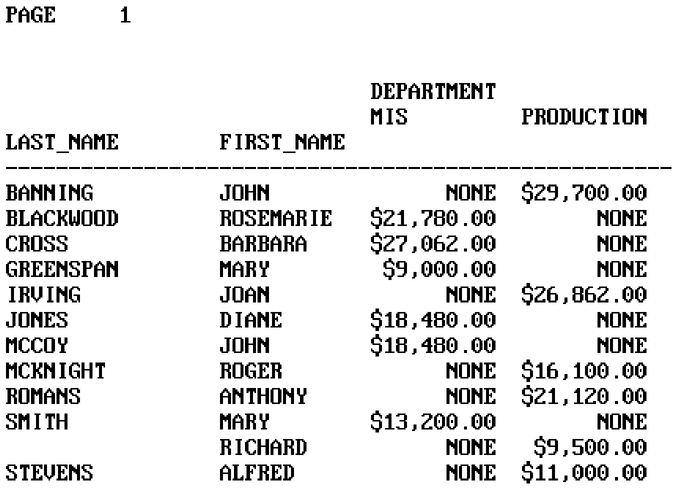
Replaces the default value.

Example Setting Parameters in a Report Request

For example, this request changes the NODATA character for missing data from a period (default) to the word NONE. No equal sign is allowed.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL
BY LAST_NAME BY FIRST_NAME
ACROSS DEPARTMENT
ON TABLE SET NODATA NONE
END
```

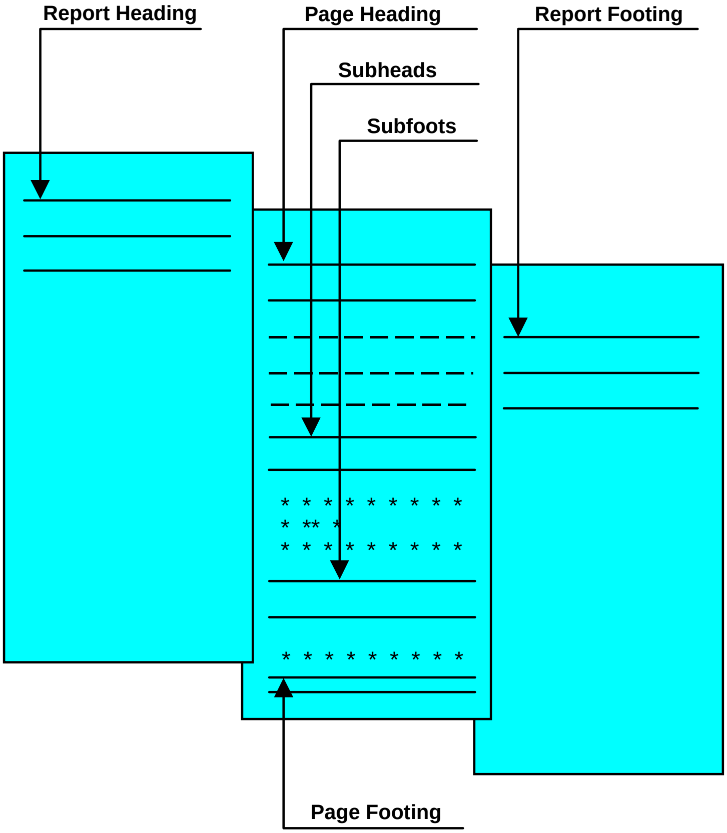
This request produces the following report:



```
PAGE     1


                                  DEPARTMENT
                                  MIS         PRODUCTION
LAST_NAME        FIRST_NAME
---------------------------------------------------------
BANNING          JOHN                   NONE  $29,700.00
BLACKWOOD        ROSEMARIE        $21,780.00        NONE
CROSS            BARBARA          $27,062.00        NONE
GREENSPAN        MARY              $9,000.00        NONE
IRVING           JOAN                   NONE  $26,862.00
JONES            DIANE            $18,480.00        NONE
MCCOY            JOHN             $18,480.00        NONE
MCKNIGHT         ROGER                  NONE  $16,100.00
ROMANS           ANTHONY                NONE  $21,120.00
SMITH            MARY             $13,200.00        NONE
                 RICHARD                NONE   $9,500.00
STEVENS          ALFRED                 NONE  $11,000.00
```

Producing Headings and Footings

You can use the variety of headings and footings to clarify the information presented in your reports. You may create up to 57 lines of headings and footings in a single report request. The following diagram illustrates the options available:



Report and Page Headings

A report heading is text that appears on the top of the first page of a report. A page heading is text that appears on the top of every page of a report. In general, the heading is composed of text that you supply in your report request, enclosed in double quotation marks.

Syntax How to Create a Report Heading

To create a report heading, the syntax is

```
ON TABLE [PAGE-BREAK AND] SUBHEAD
"text"
```

where:

`PAGE-BREAK`

Is an optional phrase that positions the report heading on a separate page, which is then followed by the first page of the report itself. If you do not use PAGE-BREAK, the report heading appears on Page 1, followed immediately by the page heading and column titles.

`text`

Is text that you supply between quotation marks that will appear as a heading. Each line of a heading can have 128 characters, and may occupy from 1 to 57 lines of the page (unless other footings and headings are used). The text must be on a line by itself and must immediately follow the SUBHEAD command.

Syntax How to Create a Page Heading

To place a heading on every page of the report, the syntax is

```
TABLE FILE filename
[HEADING [CENTER]]
"text"
```

where:

`HEADING`

Is optional if you place the text before the first display command; otherwise, it is required to identify the text as a heading. The command CENTER centers the heading over the text automatically.

`text`

Is the text placed within quotation marks that will appear on every page. The text can be split over multiple lines and must begin on the line immediately following the HEADING command.

If you supply two or more text lines between quotation marks, the lines are automatically adjusted into pairs to provide coverage across the printed page.

To position heading text, use spot markers as described in *Positioning Text* on page 10-40.

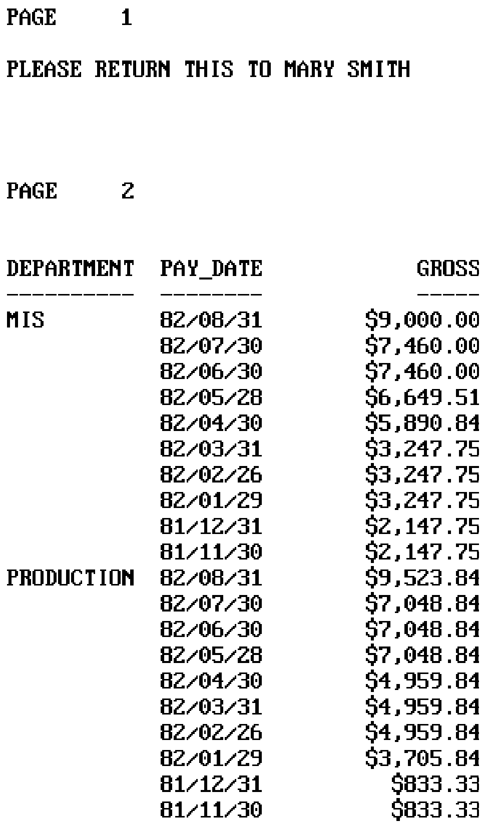
Note: A total of 6,500 characters of heading text may be supplied in a report.

Example Creating a Report Heading

For example:

```
TABLE FILE EMPLOYEE
SUM GROSS
BY DEPARTMENT BY HIGHEST PAY_DATE
ON TABLE PAGE-BREAK AND SUBHEAD
"PLEASE RETURN THIS TO MARY SMITH"
END
```

This request produces the following report (only the page preceding the body of the report has the subhead):



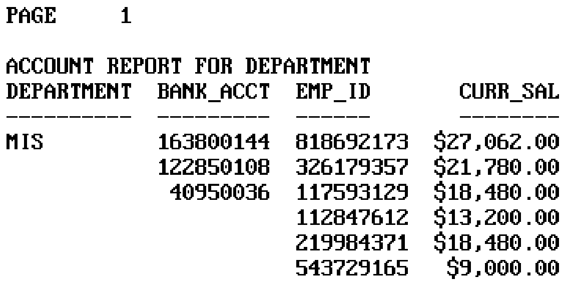
```
PAGE     1

PLEASE RETURN THIS TO MARY SMITH



PAGE     2

DEPARTMENT  PAY_DATE           GROSS
----------  --------           -----
MIS         82/08/31       $9,000.00
            82/07/30       $7,460.00
            82/06/30       $7,460.00
            82/05/28       $6,649.51
            82/04/30       $5,890.84
            82/03/31       $3,247.75
            82/02/26       $3,247.75
            82/01/29       $3,247.75
            81/12/31       $2,147.75
            81/11/30       $2,147.75
PRODUCTION  82/08/31       $9,523.84
            82/07/30       $7,048.84
            82/06/30       $7,048.84
            82/05/28       $7,048.84
            82/04/30       $4,959.84
            82/03/31       $4,959.84
            82/02/26       $4,959.84
            82/01/29       $3,705.84
            81/12/31         $833.33
            81/11/30         $833.33
```

Example Creating a Page Heading

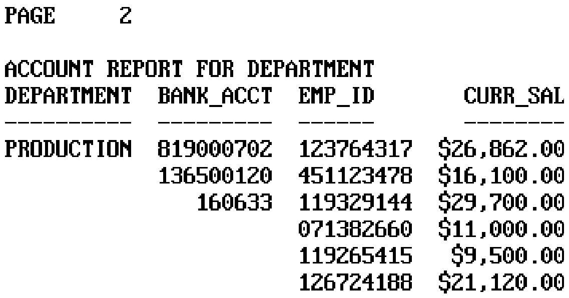
For example:

```
TABLE FILE EMPLOYEE
"ACCOUNT REPORT FOR DEPARTMENT"
PRINT CURR_SAL BY DEPARTMENT BY HIGHEST BANK_ACCT
BY EMP_ID
ON DEPARTMENT PAGE-BREAK
END
```

This request produces the following two-page report:



```
PAGE     1

ACCOUNT REPORT FOR DEPARTMENT
DEPARTMENT  BANK_ACCT  EMP_ID       CURR_SAL
----------  ---------  ------       --------
MIS         163800144  818692173  $27,062.00
            122850108  326179357  $21,780.00
             40950036  117593129  $18,480.00
                       112847612  $13,200.00
                       219984371  $18,480.00
                       543729165   $9,000.00
```



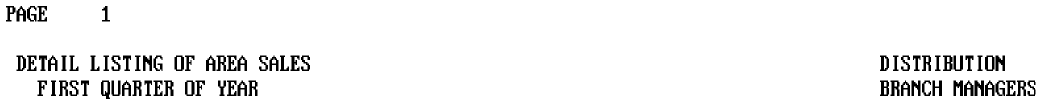
```
PAGE     2

ACCOUNT REPORT FOR DEPARTMENT
DEPARTMENT  BANK_ACCT  EMP_ID       CURR_SAL
----------  ---------  ------       --------
PRODUCTION  819000702  123764317  $26,862.00
            136500120  451123478  $16,100.00
               160633  119329144  $29,700.00
                       071382660  $11,000.00
                       119265415   $9,500.00
                       126724188  $21,120.00
```

Example Creating a Multi-Line Heading

For example:

```
TABLE FILE PROD
" DETAIL LISTING OF AREA SALES
    DISTRIBUTION"
"   FIRST QUARTER OF YEAR
    BRANCH MANAGERS"
BY PROD_CODE NOPRINT
END
```

The report heading across the top of each page appears as:



```
PAGE     1

 DETAIL LISTING OF AREA SALES                                                         DISTRIBUTION
   FIRST QUARTER OF YEAR                                                              BRANCH MANAGERS
```

DISTRIBUTION and BRANCH MANAGERS are on the far right of the report because of trailing blanks in the procedure. The open and closing quote marks indicate the length of the text. To avoid extra blanks, code <0X.

Report and Page Footings

A report footing is text that appears at the bottom of the last page of a report. A page footing is text that appears on the bottom of every page of a report. In general, the footing is composed of text that you can supply, in a report request, between quotation marks.

Syntax How to Create a Report Footing

To place a footing on the last page of the report, the syntax is

```
ON TABLE [PAGE-BREAK AND] SUBFOOT
"text"
```

where:

`PAGE-BREAK`

Is an optional phrase that positions the report footing on the last page by itself. If not used, the report footing appears as the last line on the report.

Note: If PAGE-BREAK is specified in the BY phrase and not in the ON TABLE phrase, the report footing appears as the last line on the last page of the report.

`text`

Is the text you supply in quotation marks that appears as a footing. The text begins on the line following the keyword SUBFOOT. Each line of a footing can be 128 characters in length, and may occupy from one to 57 lines of the page (unless other footings and headings are used).

Syntax How to Create a Page Footing

To display a footing on every page of a report, the syntax is

```
FOOTING [CENTER] [BOTTOM]
"text"
```

where:

`FOOTING`
: Is the keyword that identifies the text as a footing.

`CENTER`
: Centers the footing automatically.

`BOTTOM`
: Places the footing at the bottom of the page. If BOTTOM is not specified, the footing text appears two lines below the report.

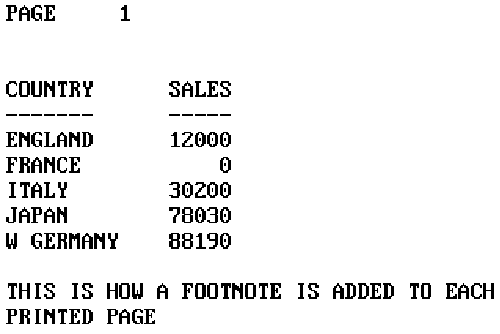
`text`
: Is the text you place within quotation marks that will appear on every page. Each line of a footing can be 128 characters in length, and may occupy from one to 57 lines of the page.

Example Creating a Page Footing

For example, this request

```
TABLE FILE CAR
WRITE SALES BY COUNTRY
FOOTING
"THIS IS HOW A FOOTNOTE IS ADDED TO EACH"
"PRINTED PAGE"
END
```

produces the following report:



```
PAGE     1


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

THIS IS HOW A FOOTNOTE IS ADDED TO EACH
PRINTED PAGE
```

Subheads

A subhead is text that can be placed before the sort field values change.

Syntax How to Create a Subhead

The syntax is

```
{ON|BY}  fieldname SUBHEAD
"text"
[WHEN expression;]
```

where:

fieldname

Is the sort field before which the text will be inserted.

text

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

WHEN *expression*

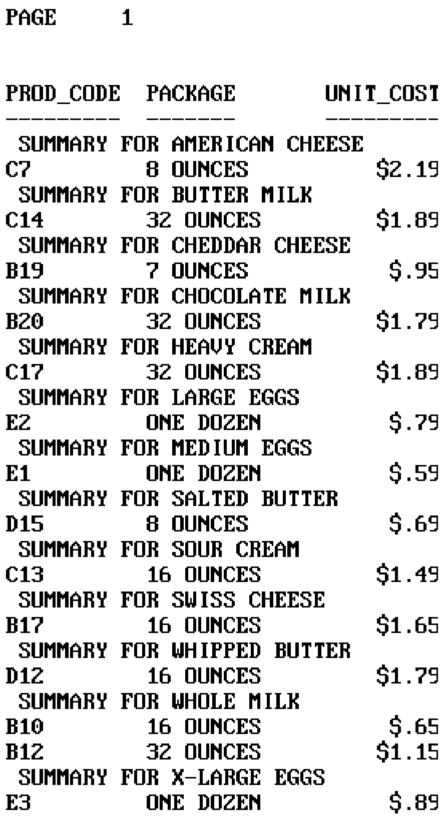
Specifies a conditional subhead in the printing of a report, as determined by a Boolean expression. Used with SUBHEAD, the WHEN clause must be placed on a line following the text you enclose in double quotation marks.

Example Using Subheads

For example, this request

```
TABLE FILE PROD
SUM PACKAGE AND UNIT_COST
BY PROD_NAME NOPRINT BY PROD_CODE
ON PROD_NAME SUBHEAD
" SUMMARY FOR <PROD_NAME"
END
```

creates the report below:



```
PAGE     1


PROD_CODE  PACKAGE       UNIT_COST
---------  -------       ---------
 SUMMARY FOR AMERICAN CHEESE
C7         8 OUNCES          $2.19
 SUMMARY FOR BUTTER MILK
C14        32 OUNCES         $1.89
 SUMMARY FOR CHEDDAR CHEESE
B19        7 OUNCES           $.95
 SUMMARY FOR CHOCOLATE MILK
B20        32 OUNCES         $1.79
 SUMMARY FOR HEAVY CREAM
C17        32 OUNCES         $1.89
 SUMMARY FOR LARGE EGGS
E2         ONE DOZEN          $.79
 SUMMARY FOR MEDIUM EGGS
E1         ONE DOZEN          $.59
 SUMMARY FOR SALTED BUTTER
D15        8 OUNCES           $.69
 SUMMARY FOR SOUR CREAM
C13        16 OUNCES         $1.49
 SUMMARY FOR SWISS CHEESE
B17        16 OUNCES         $1.65
 SUMMARY FOR WHIPPED BUTTER
D12        16 OUNCES         $1.79
 SUMMARY FOR WHOLE MILK
B10        16 OUNCES          $.65
B12        32 OUNCES         $1.15
 SUMMARY FOR X-LARGE EGGS
E3         ONE DOZEN          $.89
```

Subfoots

A subfoot is text that can be placed after the sort field values change. MULTILINES can also be used with SUBFOOT to suppress SUBFOOTs.

Syntax

How to Create Subfoots

The syntax is

```
{ON|BY}  fieldname SUBFOOT [MULTILINES]
"text"
[WHEN expression;]
```

where:

fieldname
Is the field after which the text will be inserted.

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

MULTILINES
Is used to suppress the SUBFOOT when there is only one line of output for the BY group. FOCUS also allows you to suppress grand totals using the NOTOTAL phrase as described in Chapter 7, *Including Totals and Subtotals in Your Report*.

WHEN *expression*
Specifies a conditional subfoot in the printing of a report, as determined by a Boolean expression (see *Using Direct Operators in Headings and Footings* on page 10-45). Used with SUBFOOT, WHEN must be placed on the line following the text you enclose in double quotation marks.

Reference

Usage Notes for Creating Subfoots

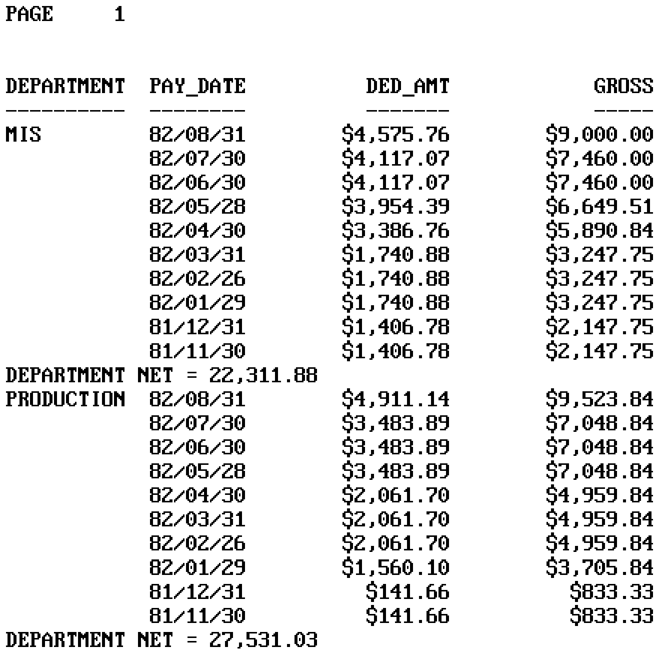
- When a SUBFOOT follows a RECAP, the default display of the RECAP values is suppressed, as it is assumed that the SUBFOOT is being used to display the RECAP. Notice that the phrase <DEPAR_NET was replaced with the actual value of DEPAR_NET.
- A SUBFOOT can also be used as a complete report request without any display command if data is embedded in the text, because fields in the text become implicit display fields. The default display command is SUM. For more information, see *Using Data in Headings and Footings* on page 10-42.
- If the report request contains the command SUM and the display field is specified in a subfoot, the value is summed. Use direct operators with fields specified in subfootings.

Example Using Subfoots

This example

```
TABLE FILE EMPLOYEE
SUM DED_AMT AND GROSS
BY DEPARTMENT BY HIGHEST PAY_DATE
ON DEPARTMENT RECAP
DEPAR_NET/D8.2=GROSS-DED_AMT;
ON DEPARTMENT SUBFOOT
"DEPARTMENT NET = <DEPAR_NET"
END
```

produces the following report:



```
PAGE     1


DEPARTMENT  PAY_DATE          DED_AMT           GROSS
----------  --------          -------           -----
MIS         82/08/31        $4,575.76       $9,000.00
            82/07/30        $4,117.07       $7,460.00
            82/06/30        $4,117.07       $7,460.00
            82/05/28        $3,954.39       $6,649.51
            82/04/30        $3,386.76       $5,890.84
            82/03/31        $1,740.88       $3,247.75
            82/02/26        $1,740.88       $3,247.75
            82/01/29        $1,740.88       $3,247.75
            81/12/31        $1,406.78       $2,147.75
            81/11/30        $1,406.78       $2,147.75
DEPARTMENT NET = 22,311.88
PRODUCTION  82/08/31        $4,911.14       $9,523.84
            82/07/30        $3,483.89       $7,048.84
            82/06/30        $3,483.89       $7,048.84
            82/05/28        $3,483.89       $7,048.84
            82/04/30        $2,061.70       $4,959.84
            82/03/31        $2,061.70       $4,959.84
            82/02/26        $2,061.70       $4,959.84
            82/01/29        $1,560.10       $3,705.84
            81/12/31          $141.66         $833.33
            81/11/30          $141.66         $833.33
DEPARTMENT NET = 27,531.03
```

Positioning Text

The positioning of text and data in headings, footings, subheads, and subfoots can be controlled by a spot marker, which identifies the column where the text should begin. A spot marker consists of a left caret (<) followed by a number indicating the absolute or relative column position. The right caret (>) is optional and can make the spot marker clearer to a reader.

The various ways spot markers can be used are illustrated in the chart below:

Marker	**Example**	**Usage**
`<`*n* `or <`*n*`>`	`<50`	The next character starts in column 50.
`<+`*n* `or <+`*n*`>`	`<+4`	The next character starts four columns from the last non-blank character.
`<-`*n* `or <-`*n*`>`	`<-1`	The next character starts one column to the left of the last character and suppresses or writes over all or part of a field.
`</`*n* `or </`*n*`>`	`</2`	Skip two lines.
`<0X or <0X>`	`<0X`	Positions the next character immediately to the right of the last character (skip zero columns). This is used when you have more than two lines between the double quotation marks in a stored procedure that make up a single line of heading, subhead, footing, or subfoot display. No spaces are inserted between the spot marker and the start of a continuation line.

Note: If you place a skip line spot marker on a line by itself, it will skip one more line than you asked for. To avoid this, put the skip line marker on the same line with additional text from the report. In addition, each field needs one space for field attributes; if a field placed with a spot marker overlaps an existing field, unpredictable results may occur.

Example Positioning Text

- To place a character in a specific column:

  ```
  "<50 SUMMARY REPORT"
  ```

  The letter S in SUMMARY will start in Position 50 of the line.

- To place a substituted value in a specific column:

  ```
  "<15 COST OF VEHICLE IS <40 <RCOST>"
  "<10 <DIVISION <30 <AREA <50 <DATE"
  ```

- To add spaces to the right of the last non-blank character:

  ```
  "DAILY REPORT <DATE <+5 <LOCATION <+5 <PRODUCT"
  ```

- To move to the left of the last non-blank character:

  ```
  "<60 CONFIDENTIAL <-40 <FIRST_NAME"
  ```

  Skipping backward may cover other text on a line. This may be useful in some cases, but in general should be avoided.

- To show four lines of heading text between double quotation marks:

  ```
  "THIS HEADING <0X
  SHOULD APPEAR <0X
  ON ONE <0X
  LINE"
  ```

  The above produces the line:

  ```
  THIS HEADING SHOULD APPEAR ON ONE LINE
  ```

- To position a long line:

  ```
  "<20 DETAIL REPORT WITH LOTS OF TEXT ON ONE LINE
  <100 EVEN THOUGH IT IS ON TWO LINES IN THE REQUEST"
  ```

- To skip multiple lines:

  ```
  "</4 THIS IS ON THE FIFTH LINE DOWN"
  ```

Using Data in Headings and Footings

You can embed the values of fields in headings, subheads, subfoots, and footings.

Syntax How to Insert Data in Headings and Footings

To put a value in one of these titles, use the following syntax

```
<fieldname

<fieldname>
```

where:

`<fieldname`
Places the data value in the heading or footing, and suppresses trailing blanks.

`<fieldname>`
Places the data value in the heading or footing, and retains trailing blanks.

Reference Usage Notes for Data in Headings and Footings

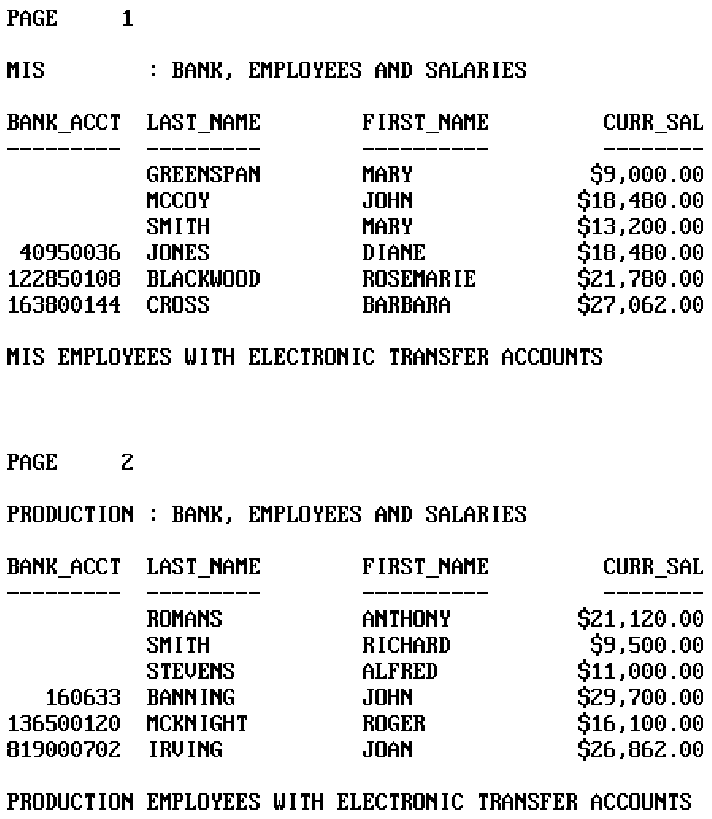
- Trailing blanks in alphanumeric fields may be omitted by using only the opening < character for data in headings. For example, if AREA is a 16-character alphanumeric field, the line is expanded by 16 characters at the point of substitution of the retrieved value. If only the opening character is used, only the non-blank characters of the particular value are substituted. For example, if <AREA retrieves the value of EAST, only four characters plus one leading blank are inserted in the line, rather than a full 16 characters which the data value could contain.
- A SUBFOOT can be used as a complete report request without any display command if data is embedded in the text, because fields in the text become implicit display fields.
- You can place page numbers in headings and footings using TABPAGENO (see *Inserting Page Numbers: TABPAGENO* on page 10-4).
- Fields in headings and footings are evaluated as if they were verb objects of the first verb. Fields in subheads and subfoots are evaluated as part of the first verb in which they are referenced. If a field is not referenced, it is evaluated as part of the last verb.

Example Using Data in a Heading

For example:

```
TABLE FILE EMPLOYEE
"<DEPARTMENT>: BANK, EMPLOYEES AND SALARIES </1"
PRINT CURR_SAL
BY DEPARTMENT NOPRINT BY BANK_ACCT
BY LAST_NAME BY FIRST_NAME
ON DEPARTMENT PAGE-BREAK
FOOTING
"<DEPARTMENT EMPLOYEES WITH ELECTRONIC TRANSFER ACCOUNTS"
END
```

This request produces the following 2-page report:



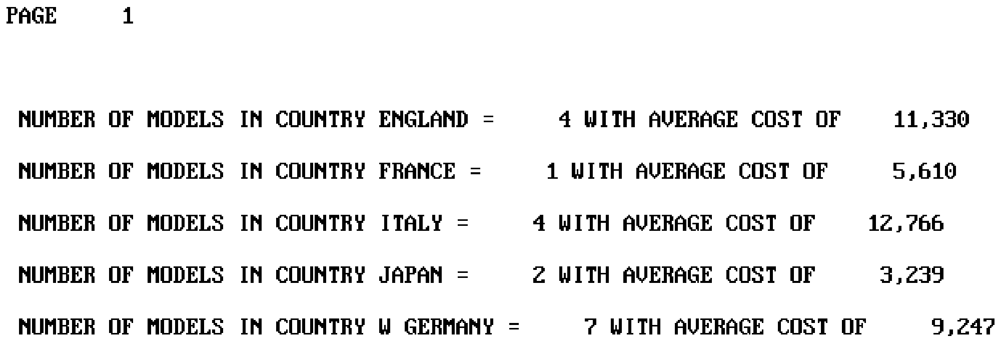
```
PAGE     1

MIS        : BANK, EMPLOYEES AND SALARIES

BANK_ACCT  LAST_NAME        FIRST_NAME         CURR_SAL
---------  ---------        ----------         --------
           GREENSPAN        MARY              $9,000.00
           MCCOY            JOHN             $18,480.00
           SMITH            MARY             $13,200.00
 40950036  JONES            DIANE            $18,480.00
122850108  BLACKWOOD        ROSEMARIE        $21,780.00
163800144  CROSS            BARBARA          $27,062.00

MIS EMPLOYEES WITH ELECTRONIC TRANSFER ACCOUNTS


PAGE     2

PRODUCTION : BANK, EMPLOYEES AND SALARIES

BANK_ACCT  LAST_NAME        FIRST_NAME         CURR_SAL
---------  ---------        ----------         --------
           ROMANS           ANTHONY          $21,120.00
           SMITH            RICHARD           $9,500.00
           STEVENS          ALFRED           $11,000.00
   160633  BANNING          JOHN             $29,700.00
136500120  MCKNIGHT         ROGER            $16,100.00
819000702  IRVING           JOAN             $26,862.00

PRODUCTION EMPLOYEES WITH ELECTRONIC TRANSFER ACCOUNTS
```

Example Using Data in a SUBFOOT

For example, this request

```
TABLE FILE CAR
BY COUNTRY NOPRINT SUBFOOT
"NUMBER OF MODELS IN COUNTRY <COUNTRY = <CNT.MODEL
WITH AVERAGE COST OF <AVE.RCOST "
END
```

creates this report:

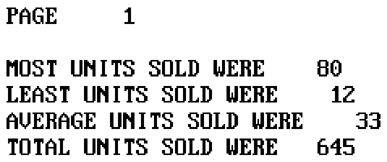


```
PAGE     1


 NUMBER OF MODELS IN COUNTRY ENGLAND =     4 WITH AVERAGE COST OF    11,330

 NUMBER OF MODELS IN COUNTRY FRANCE =     1 WITH AVERAGE COST OF     5,610

 NUMBER OF MODELS IN COUNTRY ITALY =     4 WITH AVERAGE COST OF    12,766

 NUMBER OF MODELS IN COUNTRY JAPAN =     2 WITH AVERAGE COST OF     3,239

 NUMBER OF MODELS IN COUNTRY W GERMANY =     7 WITH AVERAGE COST OF     9,247
```

Example Using Direct Operators in Headings and Footings

You can use any prefix operator in a heading or footing to perform specific operations. Consider the following examples

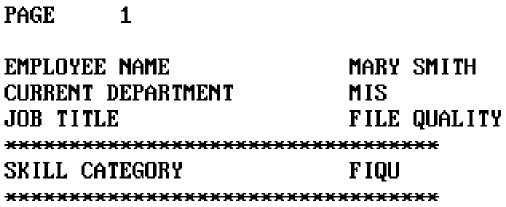
```
TABLE FILE SALES
"MOST UNITS SOLD WERE <MAX.UNIT_SOLD"
"LEAST UNITS SOLD WERE <MIN.UNIT_SOLD"
"AVERAGE UNITS SOLD WERE <AVE.UNIT_SOLD"
"TOTAL UNITS SOLD WERE <TOT.UNIT_SOLD"
END
```



```
PAGE     1

MOST UNITS SOLD WERE    80
LEAST UNITS SOLD WERE    12
AVERAGE UNITS SOLD WERE    33
TOTAL UNITS SOLD WERE   645
```

and

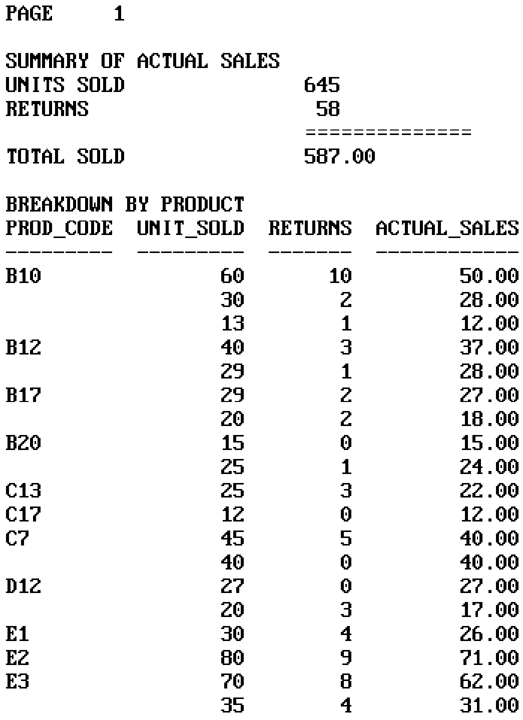
```
TABLE FILE EMPLOYEE
"EMPLOYEE NAME <FIRST_NAME <LAST_NAME"
"CURRENT DEPARTMENT   <DEPARTMENT"
"JOB TITLE <JOB_DESC"
"**********************************"
"SKILL CATEGORY       <SKILLS"
"**********************************"
" "
WHERE EMP_ID IS '112847612'
END
```



```
PAGE     1

EMPLOYEE NAME              MARY SMITH
CURRENT DEPARTMENT         MIS
JOB TITLE                  FILE QUALITY
**********************************
SKILL CATEGORY             FIQU
**********************************
```

and

```
DEFINE FILE SALES
ACTUAL_SALES/D8.2 = UNIT_SOLD-RETURNS;
%SALES/F5.1 = 100*ACTUAL_SALES/UNIT_SOLD;
END

TABLE FILE SALES
"SUMMARY OF ACTUAL SALES"
"UNITS SOLD   <TOT.UNIT_SOLD"
"RETURNS        <TOT.RETURNS"
"      =============="
"TOTAL SOLD    <TOT.ACTUAL_SALES"
" "
"BREAKDOWN BY PRODUCT"
PRINT UNIT_SOLD AND RETURNS AND ACTUAL_SALES
BY PROD_CODE
END
```



```
PAGE     1

SUMMARY OF ACTUAL SALES
UNITS SOLD               645
RETURNS                   58
                         ==============
TOTAL SOLD               587.00

BREAKDOWN BY PRODUCT
PROD_CODE  UNIT_SOLD  RETURNS  ACTUAL_SALES
---------  ---------  -------  ------------
B10               60       10         50.00
                  30        2         28.00
                  13        1         12.00
B12               40        3         37.00
                  29        1         28.00
B17               29        2         27.00
                  20        2         18.00
B20               15        0         15.00
                  25        1         24.00
C13               25        3         22.00
C17               12        0         12.00
C7                45        5         40.00
                  40        0         40.00
D12               27        0         27.00
                  20        3         17.00
E1                30        4         26.00
E2                80        9         71.00
E3                70        8         62.00
                  35        4         31.00
```

You can use the following special operators only in subheadings and subfootings:

`ST.`*`fieldname`*

Produces a subtotal value of the specified field at a sort break in the report.

`CT.`*`fieldname`*

Produces a cumulative total of the specified field.

Producing a Free-Form Report

Report requests do not have to produce a tabular display, but may consist of only the heading (as long as the heading has a data field referenced in it). If the request has no display command but there is a data field embedded in the heading, FOCUS assumes that this is a heading only request and does not print the body of the report. Any data fields referenced in the heading will be treated as if they were display fields. Their values at the time the heading is printed are what they would have been had they been mentioned as in a display command. Free-form reports are described in detail in Chapter 18, *Creating Free-Form Reports*.

Conditionally Formatting Reports With the WHEN Clause

Use the WHEN clause in a TABLE request to conditionally display summary lines and formatting options for BY fields. The expression in the WHEN clause enables you to control where options such as SUBTOTAL and SUBFOOT appear in the report.

The WHEN clause is an extension of the ON phrase and must follow the ON phrase to which it applies. One WHEN clause can be specified for each option in the ON phrase. Multiple WHEN clauses are also permitted.

Used with certain formatting options in a TABLE request, the WHEN clause controls when those formatting options are displayed. If a WHEN clause is not used, the formatting options are displayed whenever the sort field value changes.

Syntax How to Create Conditional Formatting

Syntax for the WHEN clause is

```
ON fieldname option WHEN expression[;]
```

where:

option

Is any one of the following options for the ON phrase in a TABLE:

RECAP	PAGE-BREAK	SUBHEAD
RECOMPUTE	REPAGE	SUBFOOT
SUBTOTAL	SKIP-LINE	SUB-TOTAL
UNDER-LINE	SUMMARIZE	

If the WHEN clause is used with SUBHEAD or SUBFOOT, it must be placed on the line following the text that is enclosed in double quotation marks (see *Producing Headings and Footings* on page 10-30).

`expression`

Is any Boolean expression that would be valid on the right side of a COMPUTE expression (see Chapter 8, *Using Expressions*).

Note: IF … THEN … ELSE logic is not necessary in a WHEN clause and is not supported.

All non-numeric literals in a WHEN expression must be specified with single quotation marks.

The semicolon at the end of a WHEN expression is optional and may be included for readability.

Reference Usage Notes for Conditional Formatting

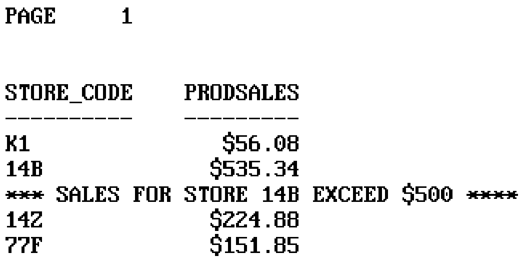
- A separate WHEN clause may be used for each option specified in an ON phrase. The ON field name phrase needs to be specified only once.
- You can use the WHEN clause to display a different SUBFOOT or SUBHEAD for each break group.
- The WHEN clause only applies to the option that immediately precedes it.
- If a WHEN clause specifies an aggregated field, the value tested is aggregated only within the break determined by the field in the corresponding ON phrase.
- In the WHEN clause for a SUBFOOT, the SUBTOTAL is calculated and evaluated. This applies to fields with prefix operators and to summed fields. For alphanumeric fields, the last value in the break group is used in the test.

Example Conditionally Formatting Reports

In the following example, the WHEN clause prints a subfoot at the break for the field STORE_CODE only when the sum of PRODSALES exceeds $500:

```
DEFINE FILE SALES
PRODSALES/D9.2M = UNIT_SOLD * RETAIL_PRICE;
END

TABLE FILE SALES
SUM PRODSALES
BY STORE_CODE
ON STORE_CODE SUBFOOT
"*** SALES FOR STORE <STORE_CODE EXCEED $500 ****"
WHEN PRODSALES GT 500
END
```

The report output looks like this:

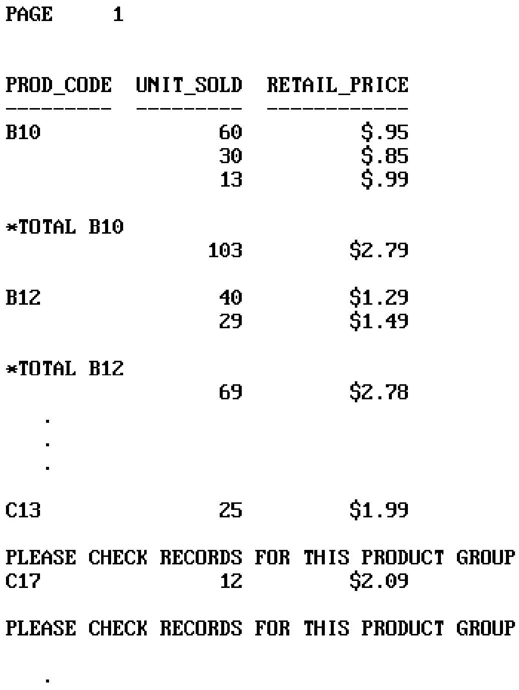


```
PAGE     1


STORE_CODE    PRODSALES
----------    ---------
K1               $56.08
14B             $535.34
*** SALES FOR STORE 14B EXCEED $500 ****
14Z             $224.88
77F             $151.85
```

Example Selectively Displaying a SUBTOTAL and SUBFOOT

For example, you can print a report that selectively displays a SUBTOTAL and a SUBFOOT:

```
TABLE FILE SALES
PRINT UNIT_SOLD RETAIL_PRICE
BY PROD_CODE
ON PROD_CODE SUBTOTAL
WHEN PROD_CODE CONTAINS 'B'      First WHEN phrase
SUBFOOT
"PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP"
WHEN PROD_CODE CONTAINS 'C'      Second WHEN phrase
END
```



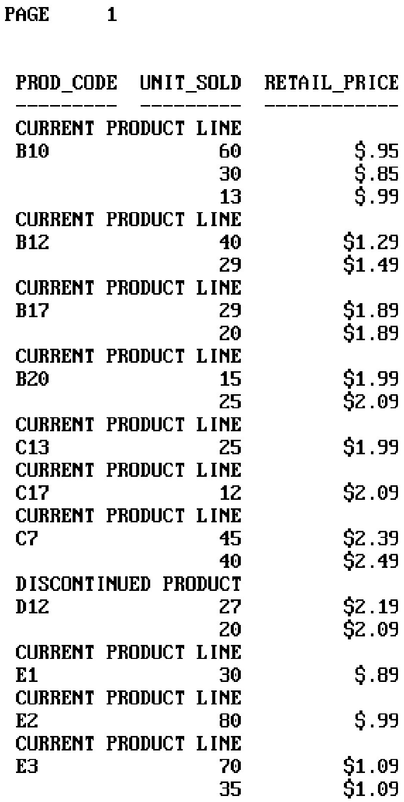
```
PAGE     1


PROD_CODE  UNIT_SOLD  RETAIL_PRICE
---------  ---------  ------------
B10               60          $.95
                  30          $.85
                  13          $.99

*TOTAL B10
                 103         $2.79

B12               40         $1.29
                  29         $1.49

*TOTAL B12
                  69         $2.78
   .
   .
   .

C13               25         $1.99

PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP
C17               12         $2.09

PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP

   .
```

Example Selectively Displaying Multiple Subheads

In the following example, a different subhead will be displayed depending on the value of the BY field. If the value of PROD_CODE contains the literal B, C, or E, the subhead CURRENT PRODUCT LINE will be displayed. If PROD_CODE contains the literal D, the subhead DISCONTINUED PRODUCT will be displayed.

```
TABLE FILE SALES
PRINT UNIT_SOLD RETAIL_PRICE
BY PROD_CODE

ON PROD_CODE
SUBHEAD
"CURRENT PRODUCT LINE"
WHEN PROD_CODE CONTAINS 'B' OR 'C' OR 'E'

SUBHEAD
"DISCONTINUED PRODUCT"
WHEN PROD_CODE CONTAINS 'D'
END
```

This produces the following report:



```
PAGE     1


 PROD_CODE  UNIT_SOLD  RETAIL_PRICE
 ---------  ---------  ------------
 CURRENT PRODUCT LINE
 B10               60          $.95
                   30          $.85
                   13          $.99
 CURRENT PRODUCT LINE
 B12               40         $1.29
                   29         $1.49
 CURRENT PRODUCT LINE
 B17               29         $1.89
                   20         $1.89
 CURRENT PRODUCT LINE
 B20               15         $1.99
                   25         $2.09
 CURRENT PRODUCT LINE
 C13               25         $1.99
 CURRENT PRODUCT LINE
 C17               12         $2.09
 CURRENT PRODUCT LINE
 C7                45         $2.39
                   40         $2.49
 DISCONTINUED PRODUCT
 D12               27         $2.19
                   20         $2.09
 CURRENT PRODUCT LINE
 E1                30          $.89
 CURRENT PRODUCT LINE
 E2                80          $.99
 CURRENT PRODUCT LINE
 E3                70         $1.09
                   35         $1.09
```

Example Selectively Displaying a Subfoot

In the following example

```
ON PROD_CODE SUBTOTAL AND SUBFOOT
"PLEASE CHECK RECORDS FOR THIS PRODUCT GROUP"
WHEN PROD_CODE CONTAINS 'B'
```

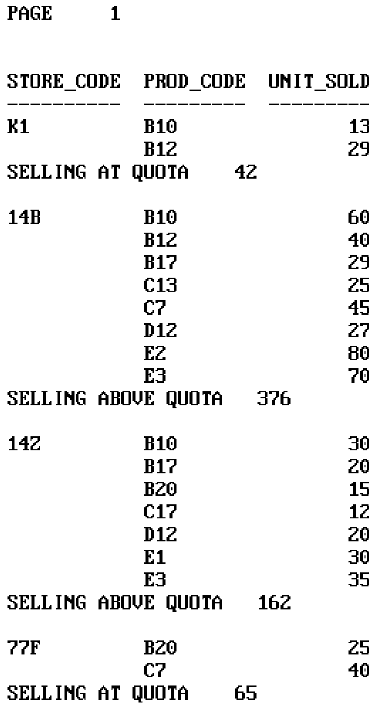
a subtotal will be calculated for each PROD_CODE, but the subfoot will be displayed only when PROD_CODE contains the literal B.

Example Using Aggregation in the WHEN Clause

For example:

```
TABLE FILE SALES
SUM UNIT_SOLD
 BY STORE_CODE BY PROD_CODE

ON STORE_CODE SUBFOOT
"SELLING ABOVE QUOTA <ST.UNIT_SOLD "
" "
WHEN UNIT_SOLD GT 100
SUBFOOT
"SELLING AT QUOTA <ST.UNIT_SOLD"
" "
WHEN UNIT_SOLD GE 40 AND UNIT_SOLD LT 100
SUBFOOT
"SELLING BELOW QUOTA <ST.UNIT_SOLD"
" "
WHEN UNIT_SOLD LT 40
END
```

This request produces the following report:



```
PAGE     1


STORE_CODE  PROD_CODE  UNIT_SOLD
----------  ---------  ---------
K1          B10               13
            B12               29
SELLING AT QUOTA    42

14B         B10               60
            B12               40
            B17               29
            C13               25
            C7                45
            D12               27
            E2                80
            E3                70
SELLING ABOVE QUOTA   376

14Z         B10               30
            B17               20
            B20               15
            C17               12
            D12               20
            E1                30
            E3                35
SELLING ABOVE QUOTA   162

77F         B20               25
            C7                40
SELLING AT QUOTA    65
```

Controlling the Display of Empty Reports

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

Syntax How to Control Empty Reports

Use this command:

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

Valid values are:

`ON`

Generates an empty report when zero records are found.

`OFF`

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

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

```
ON TABLE SET EMPTYREPORT ON
```

Reference Usage Notes for Displaying Empty Reports

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

11 Styling Reports: StyleSheets

Topics:

- Introduction to StyleSheets
- Creating a StyleSheet

With StyleSheets, you can produce visually interesting, presentation-quality reports that highlight key information. You can choose the fonts, colors, and text styles for individual report components, either unconditionally or based on data values and conditions. You can also move columns anywhere in the report and control page parameters such as margins, size, and orientation.

For example, you can make a column title 18 point Times, make data values in the same column 16 point Helvetica with different colors for values that satisfy certain conditions, and you can italicize the column total. You can also move that column anywhere in the report.

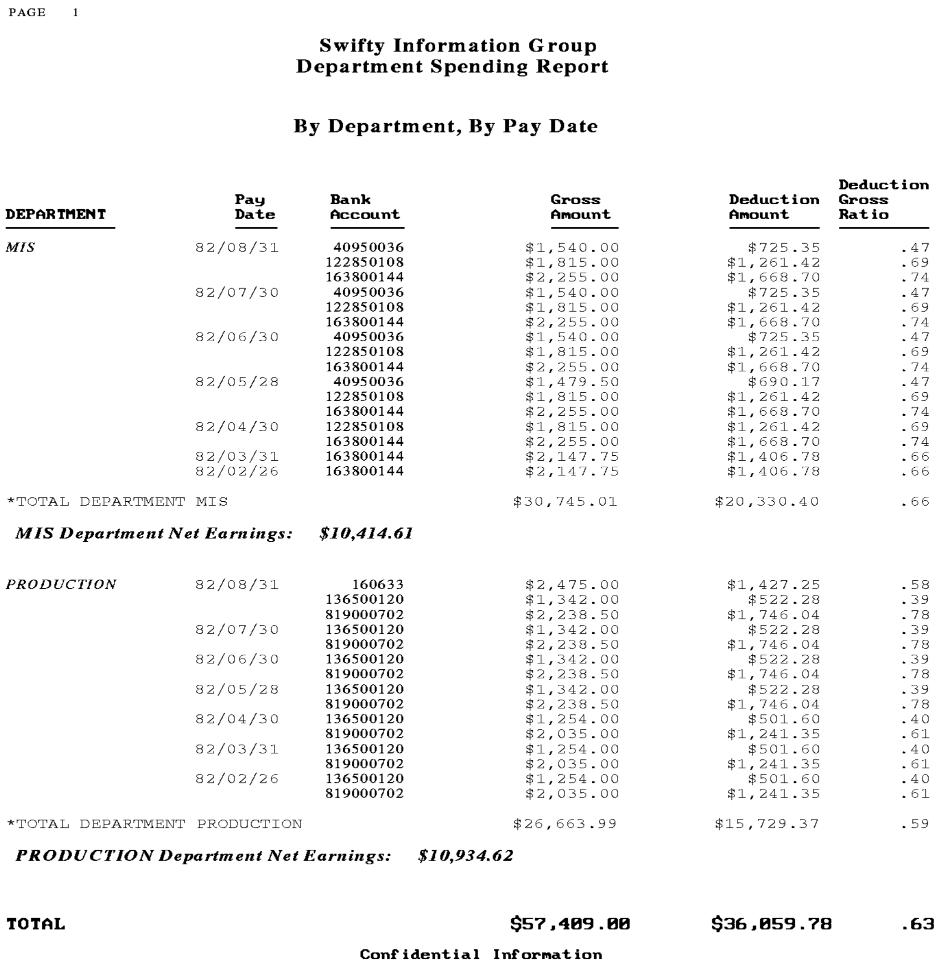
StyleSheets provide numerous options, so you can create extremely detailed formats for every line, column, or value in your report. This powerful customization and its detailed syntax makes the process of using StyleSheets seem more complicated than it actually is. In most cases, you will want to use the formatting facilities judiciously to make important information stand out. The process of using a StyleSheet consists of the following steps:

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

   Note: If the only attributes you want to change are page layout parameters, you can change the page parameters with a SET command or in a report request. *Styling the Page Layout* on page 11-12 discusses page layout parameters.

2. Activate the StyleSheet you have chosen or create a StyleSheet in a report request.
3. Create a PostScript file that contains the formatted report.
4. Print the report on a PostScript printer. This step is highly dependent on the equipment and software at your site. See your system administrator for instructions.

The following sample report shows how StyleSheets can enhance the usefulness and appearance of your reports:



```
PAGE    1

                              Swifty Information Group
                             Department Spending Report

                             By Department, By Pay Date

                                                                              Deduction
                   Pay        Bank                  Gross          Deduction  Gross
DEPARTMENT         Date       Account               Amount         Amount     Ratio

MIS                82/08/31    40950036          $1,540.00         $725.35       .47
                              122850108          $1,815.00       $1,261.42       .69
                              163800144          $2,255.00       $1,668.70       .74
                   82/07/30    40950036          $1,540.00         $725.35       .47
                              122850108          $1,815.00       $1,261.42       .69
                              163800144          $2,255.00       $1,668.70       .74
                   82/06/30    40950036          $1,540.00         $725.35       .47
                              122850108          $1,815.00       $1,261.42       .69
                              163800144          $2,255.00       $1,668.70       .74
                   82/05/28    40950036          $1,479.50         $690.17       .47
                              122850108          $1,815.00       $1,261.42       .69
                              163800144          $2,255.00       $1,668.70       .74
                   82/04/30   122850108          $1,815.00       $1,261.42       .69
                              163800144          $2,255.00       $1,668.70       .74
                   82/03/31   163800144          $2,147.75       $1,406.78       .66
                   82/02/26   163800144          $2,147.75       $1,406.78       .66

*TOTAL DEPARTMENT MIS                           $30,745.01      $20,330.40       .66

 MIS Department Net Earnings:   $10,414.61

PRODUCTION         82/08/31      160633          $2,475.00       $1,427.25       .58
                              136500120          $1,342.00         $522.28       .39
                              819000702          $2,238.50       $1,746.04       .78
                   82/07/30   136500120          $1,342.00         $522.28       .39
                              819000702          $2,238.50       $1,746.04       .78
                   82/06/30   136500120          $1,342.00         $522.28       .39
                              819000702          $2,238.50       $1,746.04       .78
                   82/05/28   136500120          $1,342.00         $522.28       .39
                              819000702          $2,238.50       $1,746.04       .78
                   82/04/30   136500120          $1,254.00         $501.60       .40
                              819000702          $2,035.00       $1,241.35       .61
                   82/03/31   136500120          $1,254.00         $501.60       .40
                              819000702          $2,035.00       $1,241.35       .61
                   82/02/26   136500120          $1,254.00         $501.60       .40
                              819000702          $2,035.00       $1,241.35       .61

*TOTAL DEPARTMENT PRODUCTION                    $26,663.99      $15,729.37       .59

 PRODUCTION Department Net Earnings:   $10,934.62


TOTAL                                           $57,409.00      $36,059.78       .63
                              Confidential Information
```

This section describes:

- Basic StyleSheet concepts, and requirements for using StyleSheets (see *Introduction to StyleSheets* on page 11-3).
- How to use an existing StyleSheet, either a custom StyleSheet or the default StyleSheet that is included with FOCUS (see *Activating an Existing StyleSheet File* on page 11-10).
- How to print styled reports (see *Printing Styled Reports* on page 11-11).
- Page layout settings (see *Styling the Page Layout* on page 11-12).

- Report components (see *Report Components* on page 11-20).
- StyleSheet file syntax and instructions for checking a StyleSheet (see *The StyleSheet File* on page 11-16).
- Style definition syntax (see *The Style Definition* on page 11-18).
- Instructions for selecting a report component, controlling column placement, and determining inheritance (see *Selecting and Manipulating Report Components* on page 11-23).
- Conditional styling with WHEN (see *Conditional Styling* on page 11-47).

Introduction to StyleSheets

This section introduces StyleSheet concepts and terminology.

What Is a StyleSheet?

A StyleSheet is a text file that describes how you want your report to look. It consists of a series of declarations that:

- Identify a report component.
- Describe the desired characteristics of that component.

You can create a StyleSheet using any text editor, including TED, the FOCUS text editor.

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

If you create a StyleSheet, you only have to define the styles of those components that you want printed differently from the default style. Any component that you do not specifically format in your StyleSheet either uses the default style or inherits a style from a higher-level component. Inheritance is discussed in *Interpreting Inheritance* on page 11-42.

When you use a StyleSheet, your page layout differs from a FOCUS report that does not use a StyleSheet. This variation is a function of the page layout parameters FOCUS uses for the two kinds of reports:

- For reports without StyleSheets, FOCUS uses the parameters LINES, PAPER, PANEL, and WIDTH to define the page layout.
- For reports with StyleSheets, FOCUS uses the parameters PAGESIZE, ORIENTATION, UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, and SQUEEZE. Any of these parameters that you do not specifically set—either in a report request, in a StyleSheet, or via a SET command—inherit default values.

What Is a Style?

A style is a description of the physical characteristics of a report component; it consists of four attributes:

- A font (typeface) such as Helvetica, Courier, or Times.
- A size for the font such as 12-point or 14-point.
- A text style or combination of text styles such as bold, italic, or bold+italic.
- A color.

When You Need to Create a StyleSheet File

You can take advantage of most StyleSheet options without ever having to create a StyleSheet file.

You can select a StyleSheet, page size, orientation, and margins at the FOCUS command level (if you want to apply them to your entire FOCUS session), or in a report request (if you want to apply them to one report).

You need to create a StyleSheet file if you wish to:

- Style report components individually.
- Apply different styles to the same component based on report values.

Requirements

To print reports using a StyleSheet file, you need:

- A PostScript printer.
- Adobe Font Metrics (AFM) files that define the measurements of characters for PostScript output.

  AFM files are distributed with FOCUS.

To use a StyleSheet file, FOCUS needs a font location file that maps the font names to the Font Metric files. If this file is unavailable and you request a StyleSheet, the following error results:

```
ERROR AT OR NEAR LINE     19 IN PROCEDURE focexec name FOCEXEC *
(FOC3222) THE FONT LOCATION FILE IS MISSING:
```

Reports that use StyleSheets can reference a maximum of 128 verb objects. If the report exceeds 128 verb objects, the following error is produced:

```
(FOC3228) STYLED REPORTS CURRENTLY LIMITED TO 128 COLUMNS.
```

Reproducing StyleSheet Examples

The examples presented in this chapter use data from the EMPLOYEE and CAR databases. In order to reproduce the examples, these sample databases must be loaded on your machine.

The sample reports use the default page size specification, LETTER, which represents 8.5 x 11 inch pages. They have been scaled down to accommodate the size of this manual.

If the fonts you have on your system do not include the ones used in the examples, substitute suitable and available fonts before you run the examples.

Required Files and DDNAMES

When you produce a report using a StyleSheet, you need the following files:

Name	**Purpose**	**MVS**	**CMS**
StyleSheet files. You can create them with a text editor (see *The StyleSheet File* on page 11-16).	Define the styles in reports.	Any member of the PDS allocated to ddname FOCSTYLE	File type FOCSTYLE, any file name
Adobe Font Metrics (AFM) files (supplied with FOCUS).	Define the measurements of characters for PostScript output.	Member names start with PS. Allocated to ddname ERRORS	File type AFM, any file name
Font location file (supplied with FOCUS).	Maps font names to the Font Metrics files.	Member PSCRIPT in the PDS allocated to ddname ERRORS.	File name PSCRIPT, File type FOCFTMAP
PostScript output files. You create these with a HOLD, SAVE, or SET command (see *Printing Styled Reports* on page 11-11).	Contain the formatted output for the PostScript printer.	Any member of the variable length PDS allocated to ddname PS.	File type PS, any file name

Comparison of Reports With and Without StyleSheets

In a non-styled FOCUS report request, you can set values for the maximum number of lines on the output page (LINES), the number of lines on the printed page (PAPER), the maximum number of characters in a report panel (PANEL), and the maximum number of characters on an output line (WIDTH). FOCUS then uses the typeface and size defined by your printer setup for all data in the report.

In a styled report, you can set margins on the top, bottom and sides of the report, you can set the page size for letters, envelopes, and many other types of paper, you can specify measurement units such as inches or points, and you can control column widths or spacing. You can also change typefaces, type size, and type style for any part of the report.

The following table compares what you can do with and without StyleSheets:

	With StyleSheets	**Without StyleSheets**
Text font	You can use different font sizes and fonts. You can selectively apply text styles such as bold or italics.	FOCUS uses the default font specified in your printer setup for the entire report.
Colors	You can select a color for the text.	The ink and paper in your printer determine the colors in your report.
Individual components	You can assign different styles to individual report components.	FOCUS uses a single style for the entire report.
Conditional styling	You can apply different styles to the same component based on report values.	Report format does not change with changes in report values.
Column widths	You can have column widths based on the column content, the field format specified in the Master File, or specify a width.	FOCUS bases column widths on the column title or the field format specified either in the Master File or the report request.

	With StyleSheets	**Without StyleSheets**
Column placement	You can specify the starting position of individual columns and arrange columns in any order regardless of the sequence established in the report request. In addition, you can indicate how much space to leave before and after individual columns.	You can specify the starting position of individual columns in the report request.
Page size	You can select from a wide range of page sizes.	You can specify the number of lines per page and characters per line within the limits of your printer setup.
Page orientation	You can select either portrait or landscape.	FOCUS uses the default orientation specified in your printer setup.
Page margins	You can specify the top, bottom, left, and right margins, measured in inches, centimeters, or points.	FOCUS uses the default page margins specified in your printer setup.

Creating a StyleSheet

There are two ways to create a StyleSheet:

- You can create a StyleSheet file in a report request. This is useful when you are applying that set of styles to only one report. See *Creating a StyleSheet Within a Report Request* on page 11-8.
- You can create a separate StyleSheet file. In order to use a StyleSheet file, you must activate it. This option is useful when you want to create a StyleSheet template that you can apply to any report. In addition, you can create a StyleSheet on one platform and then port it to and run it on other platforms. See *Activating an Existing StyleSheet File* on page 11-10.

Creating a StyleSheet Within a Report Request

You can create a StyleSheet file within your report request. This method enables you to create and maintain the styles for your report directly in the report request.

Syntax How to Create a StyleSheet in a Report Request

```
ON TABLE SET STYLE *
.
.
.
ENDSTYLE
```

where:

`STYLE *`

Indicates the beginning of an inline StyleSheet.

`ENDSTYLE`

Indicates the end of an inline StyleSheet.

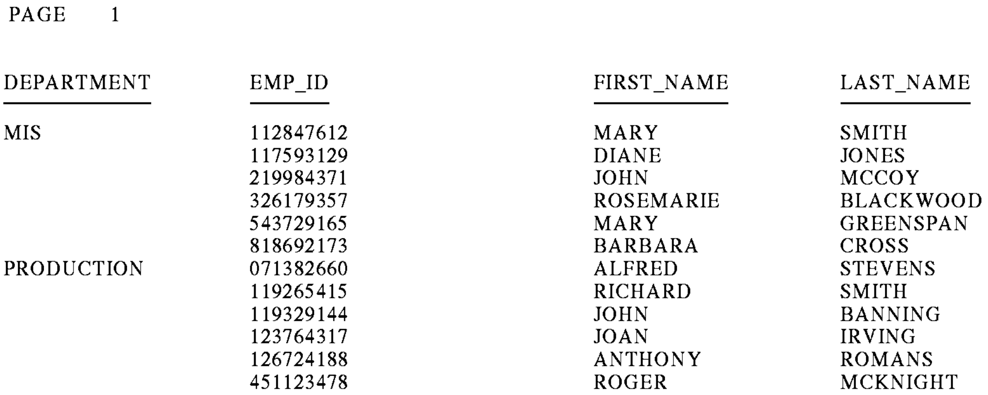
Note: You can omit the keyword ENDSTYLE, but only if it is immediately followed by the keyword END in the report request.

Example A StyleSheet Within a Report Request

In the following report request, the StyleSheet syntax appears in bold.

```
TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME
BY DEPARTMENT
ON TABLE HOLD FORMAT PS
ON TABLE SET STYLE *
TYPE=REPORT, FONT=TIMES, SIZE=10, $
TYPE=REPORT, COLUMN=EMP_ID, RIGHTGAP=1, $
ENDSTYLE
END
```

The request produces the following report:



```
PAGE     1

DEPARTMENT        EMP_ID                      FIRST_NAME        LAST_NAME

MIS               112847612                   MARY              SMITH
                  117593129                   DIANE             JONES
                  219984371                   JOHN              MCCOY
                  326179357                   ROSEMARIE         BLACKWOOD
                  543729165                   MARY              GREENSPAN
                  818692173                   BARBARA           CROSS
PRODUCTION        071382660                   ALFRED            STEVENS
                  119265415                   RICHARD           SMITH
                  119329144                   JOHN              BANNING
                  123764317                   JOAN              IRVING
                  126724188                   ANTHONY           ROMANS
                  451123478                   ROGER             MCKNIGHT
```

Activating an Existing StyleSheet File

The STYLESHEET parameter setting determines whether a report uses a custom StyleSheet. It also determines which page layout parameters are activated.

You can specify the STYLESHEET setting with a SET command if you want the setting to apply to your entire FOCUS session, or in a report request if you want the setting to apply to just that report. The syntax for the SET command is

```
SET STYLE[SHEET] = styoption
```

and the syntax in a report request is

```
TABLE FILE file
   request
ON TABLE SET STYLE = styoption
END
```

where:

styoption

Is one of the following options:

ON uses default styles. This is the default setting. With this setting in effect, FOCUS uses the page layout settings for UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, PAGESIZE, ORIENTATION, and SQUEEZE, and FOCUS ignores the settings for LINES, PAPER, PANEL, and WIDTH.

OFF uses default styles. In this case, FOCUS uses the settings for LINES, PAPER, PANEL, and WIDTH, and it ignores the settings for UNITS, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, TOPMARGIN, PAGESIZE, ORIENTATION, and SQUEEZE. The report is printed in fixed-width Courier typeface with .250-inch margins. You can use this setting to print traditional-looking FOCUS reports on PostScript printers.

Note: To disable StyleSheets entirely so that no StyleSheet is activated, use the ONLINE-FMT setting discussed in *Printing Styled Reports* on page 11-11.

stysheet is the eight-character name of a StyleSheet file. This setting activates the named StyleSheet. FOCUS uses the page layout settings for UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, PAGESIZE, ORIENTATION, and SQUEEZE, and FOCUS ignores the settings for LINES, PAPER, PANEL, and WIDTH.

Page layout settings are discussed in *Styling the Page Layout* on page 11-12, and custom StyleSheets are described in *The StyleSheet File* on page 11-16.

Printing Styled Reports

In order to print a report that was formatted using a StyleSheet, you must generate a PostScript file containing the formatted report. You must then print this file on a PostScript printer. When you view the report on your screen, you do not see the special formatting unless you have a PostScript viewer that can produce the formats on-line.

You can generate StyleSheet report output in either of two ways:

- Create a HOLD or SAVE file containing the report output in PostScript format. You can issue the HOLD or SAVE command either from the FOCUS command line or in a TABLE request. The syntax is

  ```
  [ON TABLE]  {HOLD|SAVE}   FORMAT   {PS|POSTSCRIPT}   [AS filename]
  ```

  where:

  filename

  Assigns a 1- to 8-character file name or ddname to the report saved in PostScript format. The default file name is HOLD. If a PDS is allocated to ddname PS, the output file becomes member *filename* in that PDS; otherwise, the output goes to a temporary sequential data set allocated to ddname *filename*.

  `POSTSCRIPT|PS`

  Stores the report output in a PostScript file.

- Issue the following command to select a printer driver

  ```
  SET ONLINE-FMT =  {STANDARD|POSTSCRIPT}
  ```

  where:

  `STANDARD`

  Is the default. It overrides the STYLESHEET setting and simulates standard FOCUS formatting.

  `POSTSCRIPT`

  Stores the report output in a PostScript file named PSOUT. It respects the STYLESHEET setting. PS is a synonym for POSTSCRIPT.

When you generate stylized report output that is too wide to fit in the defined print area of a single page, StyleSheet formatting divides the output across multiple pages or panels. The pages are automatically numbered with decimal notation indicating the panel number (1.1, 1.2, and so on).

Prior to generating the stylized output, FOCUS displays a FOC3218 information message indicating the total width of the report and that the resulting report output spans multiple pages.

Styling the Page Layout

When a StyleSheet is activated, FOCUS uses page parameters to format the page layout. These parameters have default values that remain in effect unless you change them in one of three ways:

- With a SET command that you issue at the FOCUS command line. Your settings remain in effect for the entire FOCUS session, unless you change them. The syntax is:

  ```
  SET parameter=value [ ,...,parameter=value ]
  ```

- With an ON TABLE phrase in a report request. These settings are in effect for the duration of the request and override values specified at the command line. The syntax is:

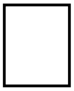
  ```
  TABLE FILE file
     request
  ON TABLE SET parameter value [ ,..., parameter value ]
  END
  ```

- In a StyleSheet. These settings are in effect whenever you activate the StyleSheet. The values specified in an activated StyleSheet override values specified at the command line or in a report request. The syntax is:

  ```
  [TYPE=REPORT] [parameter=value ,..., parameter=value]
  ```

  StyleSheet file declarations are discussed more fully in *The StyleSheet File* on page 11-16.

The parameters and values for styling a page layout are listed in the following chart:

Parameter Name (Default Value)	**Description**	**Possible Values**
ORIENTATION (PORTRAIT)	Specifies whether the report pages are long or wide.	PORTRAIT  LANDSCAPE
UNITS (INCHES)	Specifies the measurement unit for margins, gaps, and column widths.	INCHES CM (centimeters) PTS (points; one inch = 72 points; one cm = 28.35 points.) **Note:** If you change the UNITS setting, FOCUS preserves the margins by converting the numbers to the new measurement unit. For example, if a margin is 1 inch and you change the UNITS to CM, FOCUS converts the margin to 2.54 cm (the equivalent of 1 inch).
TOPMARGIN (0.25 inch)	Sets the top boundary of report contents on a page.	The desired margin value in the measurement unit specified by the UNITS parameter. Margins are measured from the edge of the physical page defined by the PAGESIZE parameter. They must account for the printable area of a page.
BOTTOMMARGIN (0.25 inch)	Sets the bottom boundary of report contents on a page.	The desired margin value in the measurement unit specified by the UNITS parameter. Margins are measured from the edge of the physical page defined by the PAGESIZE parameter. They must account for the printable area of a page.
LEFTMARGIN (0.25 inch)	Sets the left boundary of report contents on a page.	The desired margin value in the measurement unit specified by the UNITS parameter. Margins are measured from the edge of the physical page defined by the PAGESIZE parameter. They must account for the printable area of a page.

Parameter Name (Default Value)	Description	Possible Values
RIGHTMARGIN (0.25 inch)	Sets the right boundary of report contents on a page.	The desired margin value in the measurement unit specified by the UNITS parameter. Margins are measured from the edge of the physical page defined by the PAGESIZE parameter. They must account for the printable area of a page.
SQUEEZE (ON)	Determines how column widths are assigned.	ON Column widths are the larger of the widest data value or the column title. OFF Column widths are the larger of the column title or the field format in the Master File. *n* Is the actual width in UNITS. This setting is available only within a StyleSheet. If the width cannot accommodate the value, FOCUS displays the part that fits and indicates that the value is truncated with an exclamation point (!) for alphanumeric data or an asterisk (*) for numeric data.

Parameter Name (Default Value)	Description	Possible Values	
		Value	**Dimensions**
PAGESIZE (LETTER)	Specifies the page size. **Note:** If the actual paper size does not match the PAGESIZE setting, your report will either be cropped or contain extra blank space.	LETTER	8.5 x 11 in
		TABLOID	11 x 17 in
		LEDGER	17 x 11 in
		LEGAL	8.5 x 14 in
		STATEMENT	5.5 x 8.5 in
		EXECUTIVE	7.5 x 10.5 in
		A3	297 x 420 mm
		A4	210 x 297 mm
		A5	148 x 210 mm
		B4	250 x 354 mm
		B5	182 x 257 mm
		FOLIO	8.5 x 13 in
		QUARTO	215 x 275 mm
		10X14	10 x 14 in
		ENVELOPE-9	3.875 x 8.875 in
		ENVELOPE-10	4.125 x 9.5 in
		ENVELOPE-11	4.5 x 10.375 in
		ENVELOPE-12	4.5 x 11 in
		ENVELOPE-14	5 x 11.5 in
		C	17 x 22 in
		D	22 x 34 in
		E	34 x 44 in
		ENVELOPE-DL	110 x 220 mm
		ENVELOPE-C3	324 x 458 mm
		ENVELOPE-C4	229 x 324 mm
		ENVELOPE-C5	162 x 229 mm
		ENVELOPE-C6	114 x 162 mm
		ENVELOPE-C65	114 x 229 mm
		ENVELOPE-B4	250 x 353 mm
		ENVELOPE-B5	176 x 250 mm
		ENVELOPE-B6	176 x 125 mm
		ENVELOPE-ITALY	110 x 230 mm
		ENVELOPE-MONARCH	3.875 x 7.5 in
		ENVELOPE-PERSONAL	3.625 x 6.5 in
		US-STANDARD-FANFOLD	14.875 x 11 in
		GERMAN-STANDARD-FANFOLD	8.5 x 12 in
		GERMAN-LEGAL-FANFOLD	8.5 x 13 in

Displaying Current Settings: The ? STYLE Query

Use the ? STYLE query to display the current settings for the STYLESHEET parameter and all page parameters. The syntax is:

```
? STYLE
```

For example:

```
>
? style
 ONLINE-FMT STANDARD
 OFFLINE-FMT   STANDARD

 STYLESHEET:   ON

 SQUEEZE   OFF
 PAGESIZE  LETTER
 ORIENTATION   PORTRAIT
 UNITS  INCHES
 LEFTMARGIN    .250
 RIGHTMARGIN   .250
 TOPMARGIN .250
 BOTTOMMARGIN  .250
```

Note: OFFLINE-FMT is reserved for future use.

The StyleSheet File

The StyleSheet file consists of a series of declarations that describe how you want your report to look. Each declaration:

- Identifies a report component or subcomponent.
- Describes the formatting to apply to that component.
- Optionally, if the component is a heading, footing, or column, specifies a position on the page for the component.
- Optionally, specifies the distance between columns, column sequence, and column width.
- Optionally, specifies a condition that must be true in order to apply the style. This technique is called stoplighting or conditional styling.

In your StyleSheet files, include declarations for only those components whose format you want to change. Within each declaration, include only those formatting attributes that you want to change.

StyleSheet Syntax

Each declaration in a StyleSheet file consists of attribute=value pairs separated by commas and terminated with a comma and dollar sign (,$). The attributes that select a component or subcomponent must come first in each declaration. You can specify all other attributes in any order. The syntax is:

```
TYPE=value1, attribute2=value2, ... ,$
```

Note:

- You can use uppercase, lowercase, or mixed case in the StyleSheet file.
- Page layout parameters automatically apply to the whole report. Therefore, declarations that set page parameters do not require a TYPE attribute. For example, the following declarations are equivalent:

  ```
  TYPE=REPORT, ORIENTATION=LANDSCAPE ,$
  ```

  ```
  ORIENTATION=LANDSCAPE,$
  ```

  See *Styling the Page Layout* on page 11-12 for a complete description of page parameters.
- Each declaration must begin on a new line.
- A declaration can use more than one line. The terminating dollar sign indicates where the declaration ends.
- You can describe a single report element in more than one declaration.
- You can include blank spaces between the attributes, values, equal signs (=), commas, and dollar sign. You can also include blank lines. FOCUS can interpret declarations with or without blank spaces or lines.
- You can include comments, either on a declaration line after the terminating dollar sign, or on a separate comment line that begins with a dollar sign.

The attributes in the StyleSheet file identify report components, manipulate them, and define styles for formatting them. *The Style Definition* on page 11-18 discusses the style definition, *Selecting and Manipulating Report Components* on page 11-23 describes how to select and manipulate report components, and *Conditional Styling* on page 11-47 describes conditional styling.

Checking a StyleSheet

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

```
CHECK STYLE filename
```

where:

filename

Is the name of the StyleSheet file.

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

The Style Definition

A style definition consists of four attributes, separated by commas, that you can specify in any order. Any attribute that you omit from the definition retains its current setting. The four attributes are

```
FONT= {font|COURIER} ,SIZE= {n|12} ,COLOR= {color|BLACK} ,

STYLE= {[±]style [± style]|NORMAL}
```

where:

font

Is the typeface to apply. A typeface is a set of letters, numbers, and punctuation marks of a given design. COURIER is the default. Other fonts available in the font metrics files supplied with FOCUS are HELVETICA, Avant Garde Gothic, BOOKMAN, HELVETICA NARROW, NEW CENTURY SCHOOLBOOK, PALATINO, and TIMES.

You can apply fonts to all report components except underlines and skipped lines.

Embedded blanks are allowed. You can use single quotation marks for text that contains commas or where case is significant.

n

Is a positive integer. Text sizes are measured in points; the smaller the number, the smaller the type. The default size is 12 points.

color

Is one of the following: BLACK, WHITE, GREEN, RED, BLUE, MAROON, OLIVE, PURPLE, NAVY, YELLOW, TEAL, GRAY, SILVER, LIME, FUCHSIA, AQUA. The default is BLACK.

You can apply colors to all report components, except skipped lines. When colors are not available, most printers use shades of gray.

style

Is one of the following text styles:

NORMAL sets text styles to the original typeface design for the font being used. This is the default.

BOLD intensifies characters, makes them **darker**.

ITALIC slants characters and may make them more *script-like*.

OUTLINE prints only the outline of characters.

UNDERLINE prints a line under characters.

+

Combines two or more text styles, or adds a characteristic to the existing style. For example, the following adds italics to the existing style:

```
STYLE = +ITALIC
```

The following specifies bold and italics together:

```
STYLE = BOLD+ITALIC
```

-

Removes a characteristic from the existing style. For example, the following removes italics from the inherited style:

```
STYLE = -ITALIC
```

Report Components

The basic concept behind StyleSheets is that a report consists of several components, each of which has a specific name. A StyleSheet file consists of style declarations for those components whose styles you want to change, along with the formatting that you want to apply to those components. Any component that you do not specifically format in your StyleSheet either retains the default style or inherits a style from a higher-level component. Inheritance is discussed in *Interpreting Inheritance* on page 11-42.

In a StyleSheet, you identify a report component with the TYPE attribute. The following chart lists all report components:

TYPE	**Report Component**
REPORT	The entire report.
PAGENUM	Default page numbers. **Note:** Styles created for page number lines do not apply to page numbers created by the TABPAGENO variable in TABLE requests. You can format TABPAGENO page numbers by defining a style for the heading or footing that contains it.
TABHEADING	A heading on the first page of a report, generated by ON TABLE SUBHEAD.
TABFOOTING	A footing on or after the last page of a report, generated by ON TABLE SUBFOOT.
HEADING	Headings at the top of each report page.
FOOTING	Footings at the bottom of each report page.
SUBHEAD	Headings before a particular sort field, generated by ON sortfield SUBHEAD.
SUBFOOT	Footings after a particular sort field, generated by ON sortfield SUBFOOT.
DATA	Report data.
TITLE	Column titles.
ACROSSTITLE	ACROSS field names (that is, field names used in ACROSS phrases).
ACROSSVALUE	ACROSS field values (that is, values of the ACROSS field). These values become column titles in the report.
SUBTOTAL	Totals generated by SUBTOTAL, SUB-TOTAL, RECOMPUTE, and SUMMARIZE.

TYPE	Report Component
GRANDTOTAL	The last total on a report, which can either be a column total generated by COLUMN-TOTAL or a grand total generated by SUBTOTAL, SUB-TOTAL, RECOMPUTE, or SUMMARIZE.
RECAP	Lines generated by ON field name RECAP or ON field name COMPUTE.
UNDERLINE	Underlines generated by ON field name UNDER-LINE.
SKIPLINE	Skipped lines generated by ON field name SKIP-LINE.

Selecting and Manipulating Report Components on page 11-23 contains annotated report output that illustrates most components.

Within certain components, you can select specific subcomponents. For example, within a heading, you can isolate a particular line or a particular field. You identify subcomponents with selection attributes (also called qualifiers). The following chart lists the attributes you can use to select a subcomponent:

Attribute	Use with TYPE:	Selects:
COLUMN	REPORT TITLE ACROSSVALUE DATA SUBTOTAL GRANDTOTAL	One or more columns. You can identify a column by its type and/or position within the report or by its field name.
ACROSSCOLUMN	REPORT TITLE DATA SUBTOTAL GRANDTOTAL	One or more columns within ACROSS groups. You can identify an ACROSSCOLUMN by its position in its ACROSS groups or by its field name.
ACROSS	ACROSSTITLE ACROSSVALUE	An ACROSS field. You can identify an ACROSS field by its position in the set of ACROSS phrases in the report request or by its field name.
BY	SUBTOTAL RECAP SUBHEAD SUBFOOT	A subtotal line for the selected BY field. You can identify a BY field by its position in the report or by its field name.

Attribute	Use with TYPE:	Selects:

`LINE`	`TABHEADING`
	`TABFOOTING`
	`HEADING`
	`FOOTING`
	`SUBHEAD`
	`SUBFOOT`
		A line in any multi-line heading or footing. Blank and skipped lines count. You identify a line by its line number within its particular heading, footing, subheading, or subfooting.

`OBJECT`	`TABHEADING`
	`TABFOOTING`
	`HEADING`
	`FOOTING`
	`SUBHEAD`
	`SUBFOOT`
		Either text phrases (OBJECT=TEXT) or fields (OBJECT=FIELD) in any heading or footing. You can qualify this value with a LINE and/or ITEM attribute.

`ITEM`	`TABHEADING`
	`TABFOOTING`
	`HEADING`
	`FOOTING`
	`SUBHEAD`
	`SUBFOOT`
		A particular text phrase or field in a heading or footing. You identify a phrase or field by its position on its line in the heading or footing. You can use the LINE and/or OBJECT attributes to select a specific text phrase or field on a specific line.

For example, to choose the third column for the entire report, use the parameters:

- TYPE=REPORT
- COLUMN=3

Complete syntax definitions and examples are included in *Selecting and Manipulating Report Components* on page 11-23.

Selecting and Manipulating Report Components

This section describes how you select a specific component or subcomponent. It also explains how to position and manipulate columns and how to determine inheritance based on the StyleSheet hierarchy.

Selecting the Entire Report

To select the entire report, use the syntax:

```
TYPE = REPORT
```

Selecting Page Numbers

To select the default page numbers FOCUS supplies, use the syntax:

```
TYPE = PAGENUM
```

Note: Page numbers created by the TABPAGENO variable in a TABLE request are a subcomponent of the heading or footing that contains them. You can select these page numbers by selecting the heading or footing (see *Selecting Headings and Footings* on page 11-26).

Selecting Underlines

The only style attribute you can vary for an underline is the color. To select underlines for formatting, use the syntax:

```
TYPE = UNDERLINE
```

Selecting Skipped Lines

You can change the size (in points) of skipped lines. To select them for formatting, use the syntax:

```
TYPE = SKIPLINE
```

Selecting Report Data

To select the data values printed in the report, the syntax is:

```
TYPE = DATA
```

You can also select data in specific columns. See *Selecting Report Columns* on page 11-33 for details.

Selecting Column Titles

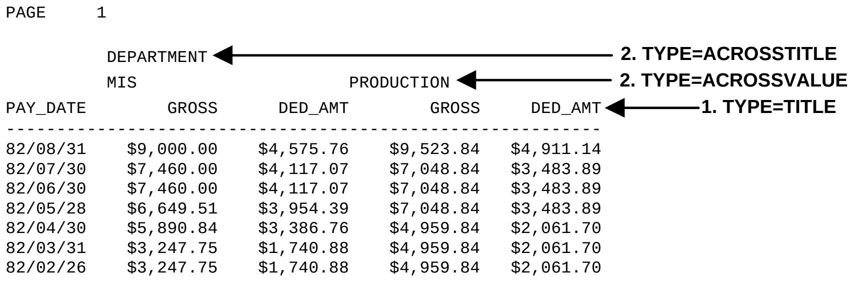
StyleSheet declarations can distinguish between three types of column titles:

TYPE	Definition
`TITLE`	This title is generated by a display command; for example, the following command displays a column titled COUNTRY: `PRINT COUNTRY`
`ACROSSTITLE`	This title consists of the field name from an ACROSS phrase in the request; for example, the following phrase displays an ACROSS group titled COUNTRY: `ACROSS COUNTRY`
`ACROSSVALUE`	This title consists of one of the values of a field referenced in an ACROSS phrase; for example, the phrase ACROSS COUNTRY displays a column titled: `LONDON`

The following request demonstrates each type of title. The numbers on the left refer to the annotated report output that follows the request:

```
    TABLE FILE EMPLOYEE
1.  SUM GROSS DED_AMT
2.  ACROSS DEPARTMENT
1.  BY HIGHEST PAY_DATE AS 'PAY DATE'
    WHERE PAY_DATE FROM 820131 TO 821231
    END
```

The numbers on the report output indicate which statements in the request produced each type of title:



```
PAGE     1

          DEPARTMENT
          MIS                        PRODUCTION
PAY_DATE        GROSS      DED_AMT        GROSS     DED_AMT
-----------------------------------------------------------
82/08/31    $9,000.00    $4,575.76    $9,523.84   $4,911.14
82/07/30    $7,460.00    $4,117.07    $7,048.84   $3,483.89
82/06/30    $7,460.00    $4,117.07    $7,048.84   $3,483.89
82/05/28    $6,649.51    $3,954.39    $7,048.84   $3,483.89
82/04/30    $5,890.84    $3,386.76    $4,959.84   $2,061.70
82/03/31    $3,247.75    $1,740.88    $4,959.84   $2,061.70
82/02/26    $3,247.75    $1,740.88    $4,959.84   $2,061.70
```

2. TYPE=ACROSSTITLE

2. TYPE=ACROSSVALUE

1. TYPE=TITLE

To select a type of title to format, use one of the following:

```
TYPE = TITLE
TYPE = ACROSSTITLE
TYPE = ACROSSVALUE
```

You can also select titles in specific columns. See *Selecting Report Columns* on page 11-33 for details.

Selecting Column Totals

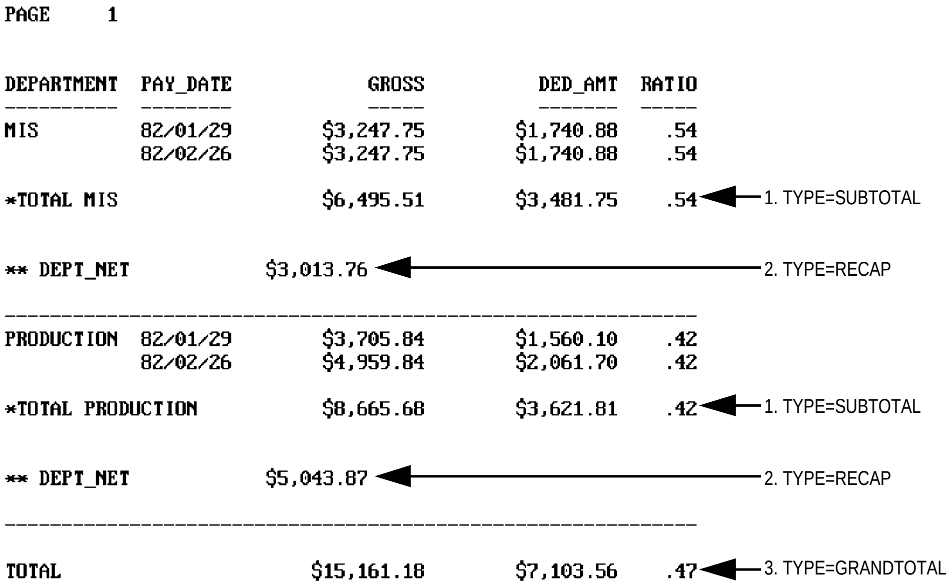
StyleSheet declarations can distinguish between three types of totals:

TYPE	Definition
GRANDTOTAL	The last total displayed in a report.
SUBTOTAL	Column totals or subtotals.
RECAP	Totals generated by RECAP or ON field COMPUTE calculations.

The following request demonstrates each type of total. The numbers on the left refer to the annotated report output that follows the request:

```
    TABLE FILE EMPLOYEE
    SUM GROSS DED_AMT AND
    COMPUTE DG_RATIO/F4.2=DED_AMT/GROSS; AS 'RATIO'
    BY DEPARTMENT
    BY PAY_DATE
1.  ON DEPARTMENT RECOMPUTE UNDER-LINE
2.  ON DEPARTMENT RECAP DEPT_NET/D8.2M=GROSS-DED_AMT;
3.  ON TABLE COLUMN-TOTAL
    WHERE PAY_DATE FROM 820101 TO 820301
    END
```

The numbers on the report output indicate which statements in the request produced each type of total:



```
PAGE     1


DEPARTMENT  PAY_DATE            GROSS          DED_AMT  RATIO
----------  --------            -----          -------  -----
MIS         82/01/29        $3,247.75        $1,740.88    .54
            82/02/26        $3,247.75        $1,740.88    .54

*TOTAL MIS                  $6,495.51        $3,481.75    .54


** DEPT_NET            $3,013.76


-------------------------------------------------------------
PRODUCTION  82/01/29        $3,705.84        $1,560.10    .42
            82/02/26        $4,959.84        $2,061.70    .42

*TOTAL PRODUCTION           $8,665.68        $3,621.81    .42


** DEPT_NET            $5,043.87


-------------------------------------------------------------

TOTAL                      $15,161.18        $7,103.56    .47
```

To select a type of total to format, the syntax is:

```
TYPE = tottype
```

where:

tottype

Can be SUBTOTAL, RECAP or GRANDTOTAL.

You can select a specific subtotal or recap line by identifying the sort field that generated it. The syntax is

```
TYPE = {SUBTOTAL|RECAP}   [,BY =   {Bn|fieldname}]
```

where:

B*n*

Identifies a BY field by its position in the report; *n* is a positive integer that counts all BY fields in the report from left to right, including NOPRINT BY fields.

fieldname

Identifies a BY field by its field name.

If a field appears in the report more than once, *fieldname*(*n*) selects the nth occurrence, and *fieldname*(*) selects all occurrences of the field.

You can also select totals in specific columns. See *Selecting Report Columns* on page 11-33 for details.

Selecting Headings and Footings

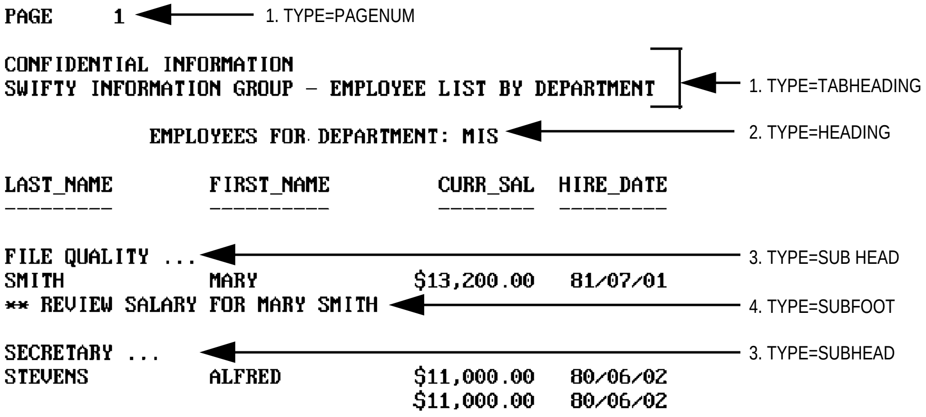
StyleSheet declarations can select the following types of headings and footings:

TYPE	**Definition**
`TABHEADING`	The report heading.
`TABFOOTING`	The report footing.
`HEADING`	The page heading.
`FOOTING`	The page footing.
`SUBHEAD`	A sort heading.
`SUBFOOT`	A sort footing.

The following request demonstrates most of the heading and footing report components. The numbers on the left refer to the annotated report output that follows the request:

```
    TABLE FILE EMPLOYEE
1.  ON TABLE SUBHEAD
1.  "CONFIDENTIAL INFORMATION"
1.  "SWIFTY INFORMATION GROUP - EMPLOYEE LIST BY DEPARTMENT </1"
2.  HEADING CENTER
2.  "EMPLOYEES FOR DEPARTMENT: <DEPARTMENT </1"
    PRINT CURR_SAL HIRE_DATE
    BY DEPARTMENT NOPRINT
3.  BY JOB_DESC NOPRINT SUBHEAD
3.  "</1 <JOB_DESC ..."
    BY LAST_NAME
4.  BY FIRST_NAME SUBFOOT
4.  "** REVIEW SALARY FOR <FIRST <LAST"
    WHEN CURR_SAL LT 18000
    WHERE RECORDLIMIT EQ 3
    END
```

The numbers on the report output indicate which statements in the request produced each type of heading or footing:



```
PAGE     1

CONFIDENTIAL INFORMATION
SWIFTY INFORMATION GROUP - EMPLOYEE LIST BY DEPARTMENT

           EMPLOYEES FOR DEPARTMENT: MIS

LAST_NAME       FIRST_NAME         CURR_SAL  HIRE_DATE
---------       ----------         --------  ---------

FILE QUALITY ...
SMITH           MARY             $13,200.00   81/07/01
** REVIEW SALARY FOR MARY SMITH

SECRETARY ...
STEVENS         ALFRED           $11,000.00   80/06/02
                                 $11,000.00   80/06/02
```

1. TYPE=PAGENUM

1. TYPE=TABHEADING

2. TYPE=HEADING

3. TYPE=SUB HEAD

4. TYPE=SUBFOOT

3. TYPE=SUBHEAD

The following syntax selects a type of heading or footing:

```
TYPE = headtype
```

where:

headtype

Can be one of the following: TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD or SUBFOOT.

You can select a specific SUBHEAD or SUBFOOT by identifying the sort field that generated it. The syntax is

```
TYPE = {SUBHEAD|SUBFOOT}   [,BY =   {Bn|fieldname}]
```

where:

B*n*

Identifies a BY field by its position in the report; *n* is a positive integer that counts all BY fields in the report from left to right, including NOPRINT BY fields.

fieldname

Identifies a BY field by its field name.

If a field appears in the report more than once, *fieldname*(*n*) selects the nth occurrence, and *fieldname*(*) selects all occurrences of the field.

In addition to the heading or footing text, headings and footings can include database fields, DEFINE fields, and page numbers created by the TABPAGENO field. You can select specific elements of a heading or footing with the following selection attributes (qualifiers):

Attribute	**Definition**
`LINE`	Selects a particular line in a multi-line heading or footing.
`OBJECT`	Selects a type of subcomponent: text phrases or embedded fields.
`ITEM`	Selects a particular text phrase or field.

The syntax for selecting a subcomponent of a heading or footing is

```
TYPE = {headtype [BY={Bn|fieldname}][LINE=n][ ,OBJECT={TEXT|FIELD}][ITEM=m]
```

where:

headtype

Can be one of the following: TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD or SUBFOOT.

n

Selects a specific line within a multi-line heading or footing, counting blank lines specified by “ “ and lines skipped with </*n*. For example:

```
    HEADING
1.  "The <CAR <MODEL is made in <COUNTRY and"
2.  "sells for $<RCOST (as of &DATE)."
3.  " "
4.  "Number of seats: <SEATS"
```

`TEXT|FIELD`

Selects either the text phrases or the embedded fields in a heading or footing. FIELD selects only data source fields and DEFINE fields, not amper variables. For example, in the following HEADING phrase, &DATE is TEXT; DEPT is a FIELD:

```
HEADING CENTER
"Report created on &DATE for <DEPT"
```

You can use the OBJECT attribute with LINE and/or ITEM attributes to select specific text phrases or fields.

Note: Spot markers split headings and footings into multiple parts. You can deliberately split headings and footings using <+0>.

m

Selects a part of a heading or footing by its position in the heading or footing:

- If used without an OBJECT attribute, ITEM counts all text phrases and fields, starting from 1 on each line. The following selects the second item on the line, regardless of whether that item is a text phrase or a field:

  ```
  ITEM=2
  ```

- In a heading or footing with multiple text phrases, you can select a specific phrase. Count text phrases from left to right starting from 1 on each line. Text phrases are delimited by embedded fields or spot markers. The following selects the third text phrase:

  ```
  ITEM=3, OBJECT=TEXT
  ```

- In a heading or footing with multiple fields, you can select a specific field. Count fields from left to right starting from 1 on each line. The following selects the second field:

  ```
  ITEM=2, OBJECT=FIELD
  ```

- If the whole heading or footing is text with no embedded fields, the text counts as one item. You can break it into multiple items with spot markers. Consider the following HEADING phrase:

  ```
  HEADING
  "Report Created on <+0 &DATE <+0 for <DEPT"
  ```

  You can select the amper variable, &DATE, with

  ```
  ITEM=2, OBJECT=TEXT
  ```

`BY`

Is described *Selecting Headings and Footings* on page 11-26.

The following table illustrates selection attributes:

Attribute	**Definition**
`OBJECT=TEXT`	Selects all text in a heading or footing for formatting. Use the OBJECT=TEXT attribute by itself if you want all text phrases in a heading or footing to have the same style.
`OBJECT=FIELD`	Selects all embedded fields in a heading or footing for formatting. Use the OBJECT=FIELD attribute by itself if you want all fields in a heading or footing to have the same style.
`OBJECT=TEXT, ITEM=`*n*	Selects a specific text phrase for formatting. Text phrases are counted from left to right in a heading or footing line.
`OBJECT=FIELD, ITEM=`*n*	Selects a specific field for formatting. Fields are counted from left to right in a heading or footing line.
`LINE=`*n*`, OBJECT=TEXT`	Selects all text phrases on a specified heading or footing line.
`LINE=`*n*`, OBJECT=FIELD`	Selects all embedded fields on a specified heading or footing line.
`LINE=`*n*`, ITEM=`*m*	Selects the mth item on the nth line of a heading or footing, regardless of whether this item is a text phrase or an embedded field.
`LINE=`*n*`, OBJECT=TEXT, ITEM=`*m*	Selects the mth text phrase on the nth line of a heading or footing.
`LINE=`*n*`, OBJECT=FIELD, ITEM=`*m*	Selects the mth field on the nth line of a heading or footing.

The key to defining styles for subcomponents within a report heading or footing is knowing how to count the lines, text phrases, and fields within a heading or footing.

Example Styling Heading and Footing Subcomponents

This example illustrates how to count and build styles for one or more lines, fields, and text phrases in a heading or footing:

```
    TABLE FILE CAR
    HEADING
1.  "The <CAR <MODEL is made in <COUNTRY and"
2.  "sells for $<RETAIL_COST (as of &DATE)."
    " "
3.  "Number of Seats: <SEATS Weight: <WEIGHT "
    " "
    " "
    PRINT BODYTYPE NOPRINT
    FOOTING
4.  "Supplied by"
    " "
    " The National Automobile Statistics Organization"
    " "
    WHERE COUNTRY EQ 'ENGLAND';
    ON TABLE SET STYLE HEAD
    ON TABLE HOLD AS HEAD FORMAT PS
    END
```

1. The first line of the report heading consists of three embedded fields (CAR, MODEL, and COUNTRY), and three text phrases (‘The ’, ‘ is made in ’, and ‘ and’).

The following declaration defines a style for all embedded fields in the first line:

```
TYPE=HEADING, LINE=1, OBJECT=FIELD, FONT=TIMES, SIZE=16,
STYLE=NORMAL, COLOR=BLACK ,$
```

The next declaration defines a style for the second field, MODEL, in the first line:

```
TYPE=HEADING, LINE=1, OBJECT=FIELD, ITEM=2, FONT=TIMES, SIZE=10,STYLE=BOLD,
COLOR=BLACK,$
```

2. The second line of the report heading consists of one field (RETAIL_COST), and two text phrases (‘sells for $’ and ‘ (as of &DATE). ’).

The following declaration defines a style for the field RETAIL_COST:

```
TYPE=HEADING, LINE=2, OBJECT=FIELD, FONT=TIMES, SIZE=10, STYLE=BOLD,
COLOR=BLACK,$
```

The next declaration defines a style for the two text phrases:

```
TYPE=HEADING, LINE=2, OBJECT=TEXT, FONT=TIMES, SIZE=10, STYLE=ITALIC,
COLOR=BLACK,$
```

The following declaration defines a style for the second text phrase (‘(as of &DATE).’):

```
TYPE=HEADING, LINE=2, OBJECT=TEXT, ITEM=2, FONT=TIMES, SIZE=10,STYLE=
UNDERLINE, COLOR=BLACK,$
```

3. This is the fourth line of the report heading, because “ “ in the previous line of the heading specifies a blank line, and blank lines count. It consists of two fields (SEATS and WEIGHT) and two text phrases (‘Number of Seats:’ and ‘Weight:’).

   The following declaration defines a style for the first field, SEATS:

   ```
   TYPE=HEADING, LINE=4, OBJECT=FIELD, ITEM=1, FONT=HELVETICA,
   SIZE=10,STYLE=NORMAL,$
   ```

   This declaration enhances the word ‘Weight:’ in the fourth line of the report heading:

   ```
   TYPE =HEADING, OBJECT=TEXT, LINE=4, ITEM=2, SIZE=16,$
   ```

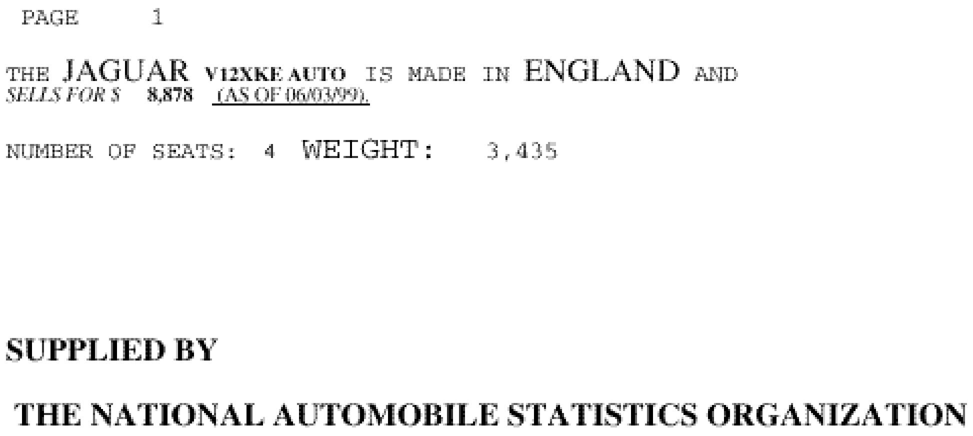
4. The four-line footing consists of two text lines (‘Supplied by’ and ‘The National Automobile Statistics Organization’).

   The following declaration defines a style for both text lines:

   ```
   TYPE=FOOTING, ITEM=1, FONT=TIMES, SIZE=16, STYLE=BOLD,$
   TYPE=FOOTING, LINE=3, FONT=TIMES, STYLE=BOLD,$
   ```

   Qualifiers are not required since there are no other sections or details (such as embedded fields) in this footing.

The StyleSheet named HEAD, consisting of these declarations, produces the following report:



```
 PAGE      1

THE JAGUAR V12XKE AUTO  IS MADE IN ENGLAND AND
SELLS FOR $    8,878  (AS OF 06/03/99).

NUMBER OF SEATS:  4  WEIGHT:    3,435




SUPPLIED BY

 THE NATIONAL AUTOMOBILE STATISTICS ORGANIZATION
```

Selecting Report Columns

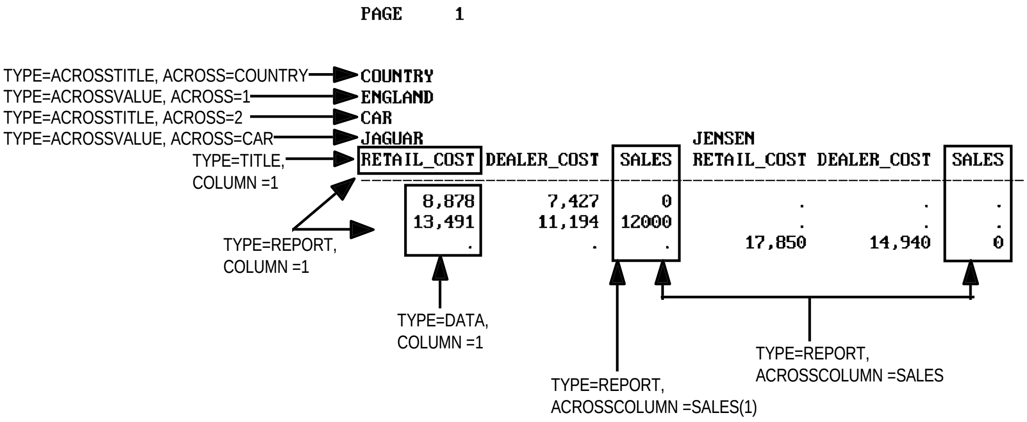
The following qualifiers select columns:

Qualifier	Selects the following subcomponent:
COLUMN	Columns in the report as a whole.
ACROSSCOLUMN	Columns in ACROSS groups.
ACROSS	A specific ACROSS field in a request with multiple ACROSS phrases.

Consider the following request that includes two ACROSS phrases (ACROSS COUNTRY and ACROSS CAR):

```
TABLE FILE CAR
PRINT RETAIL_COST DEALER_COST SALES
ACROSS COUNTRY
ACROSS CAR
WHERE COUNTRY EQ 'ENGLAND'
WHERE CAR EQ 'JAGUAR' OR 'JENSEN'
END
```

Since column attributes are subcomponents, you must use them in conjunction with a TYPE attribute that selects a report component. A subcomponent designates different parts of the report output depending on the TYPE attribute you specify with it; also, you can refer to the same column in several different ways. Some examples are marked on the following report output:



```
                    PAGE     1

                    COUNTRY
                    ENGLAND
                    CAR
                    JAGUAR                              JENSEN
                    RETAIL_COST DEALER_COST  SALES      RETAIL_COST DEALER_COST  SALES
                    -------------------------------------------------------------------
                          8,878       7,427      0                .           .      .
                         13,491      11,194  12000                .           .      .
                              .           .      .           17,850      14,940      0
```

TYPE=ACROSSTITLE, ACROSS=COUNTRY
TYPE=ACROSSVALUE, ACROSS=1
TYPE=ACROSSTITLE, ACROSS=2
TYPE=ACROSSVALUE, ACROSS=CAR
TYPE=TITLE, COLUMN =1
TYPE=REPORT, COLUMN =1
TYPE=DATA, COLUMN =1
TYPE=REPORT, ACROSSCOLUMN =SALES(1)
TYPE=REPORT, ACROSSCOLUMN =SALES

The syntax is

```
TYPE = coltype [COLUMN={colnotation | ROWTOTAL[(n)|fieldname] }]
```

or

```
TYPE = acrosscol [ACROSSCOLUMN = {acrosnotation}]
```

or

```
TYPE = {ACROSSVALUE|ACROSSTITLE}  [ACROSS = {n|fieldname}]
```

where:

coltype

Can be REPORT, TITLE, ACROSSVALUE, DATA, SUBTOTAL, or GRANDTOTAL.

colnotation

Can be *n*, P*n*, C*n*, B*n*, or *fieldname*[(*n*)].

acrosscol

Can be REPORT, TITLE, DATA, SUBTOTAL, or GRANDTOTAL.

acrosnotation

Can be *n*, P*n*, *fieldname*[(*n*)].

n

Is a positive integer that identifies a column by its position as follows:

Qualifier	**Identifies position in . . .**
`COLUMN`	The report. Count ACROSS fields, BY fields, NOPRINT fields, display fields, and ROW-TOTAL fields from left to right.
`ACROSSCOLUMN`	ACROSS groups counting from left to right.
`ACROSS`	The request. Count ACROSS phrases from top to bottom.

`P`*`n`*

Is the same as *n*, except that it does not count NOPRINT fields.

`C`*`n`*

Identifies a verb object column by its position from left to right in the report. Counts all fields, including NOPRINT fields, but excludes BY fields.

Note: C* selects all verb object columns in a report.

`B`*`n`*

Identifies a BY field by its position from left to right in the report. Counts only BY fields, including NOPRINT BY fields.

Note: B* selects all BY fields in a report.

`fieldname`

Identifies a subcomponent by its field name as follows:

When used with:	**Identifies:**
`COLUMN`	A column. When a field appears more than once, use *fieldname*(*n*) to select a particular occurrence, or *fieldname*(*) to select all occurrences. (Note that *fieldname* is equivalent to *fieldname*(1)).
`ACROSSCOLUMN`	A column in ACROSS groups. When a field appears more than once, use *fieldname*(*n*) to select a particular occurrence.
`ACROSS`	A set of ACROSS titles or data.

`ROWTOTAL`

Identifies a row total column generated by ROW-TOTAL. When used with ACROSS, ROW-TOTAL may generate multiple total columns. ROWTOTAL(*n*) selects a particular total column. ROWTOTAL(*fieldname*) selects the row total column for a particular field. ROWTOTAL(*) selects all row total columns in the report.

Example Styling With ACROSS Phrases

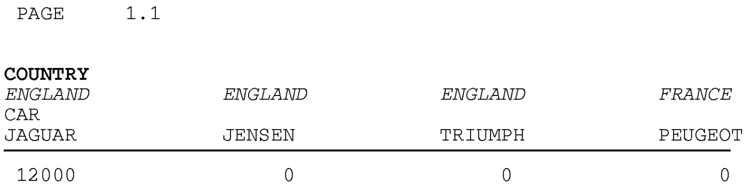
The following report request contains two ACROSS phrases:

```
TABLE FILE CAR
SUM SALES
ACROSS COUNTRY
ACROSS CAR
ON TABLE SET STYLE SAMPLE3
ON TABLE HOLD AS SAMPLE3 FORMAT PS
END
```

Consider the SAMPLE3 StyleSheet:

```
TOPMARGIN =   0.125 ,$
LEFTMARGIN =  0.125 ,$
RIGHTMARGIN = 0.125 ,$
TYPE=ACROSSTITLE, ACROSS=COUNTRY, STYLE=BOLD   ,$
TYPE=ACROSSVALUE, ACROSS=COUNTRY, STYLE=ITALIC ,$
```

The report request and the StyleSheet produce the following report:



```
 PAGE     1.1

COUNTRY
ENGLAND             ENGLAND             ENGLAND             FRANCE
CAR
JAGUAR              JENSEN              TRIUMPH             PEUGEOT
--------------------------------------------------------------------
 12000                    0                   0                   0
```

Note:

- The word COUNTRY in the report is the title produced by the phrase ACROSS COUNTRY in the request. You select it with TYPE=ACROSSTITLE, ACROSS=COUNTRY.
- The names of the countries are the values of the COUNTRY field produced by the phrase ACROSS COUNTRY in the request. You select them with TYPE=ACROSSVALUE, ACROSS=COUNTRY.
- If ACROSS COUNTRY were the only ACROSS phrase in the request, you could omit the ACROSS qualifiers in the StyleSheet.

Positioning Headings, Footings, and Columns

The POSITION attribute defines a starting position for a column, a heading, or a footing.

The syntax for positioning a column is

```
TYPE=REPORT, COLUMN = colnotation, POSITION =  {n|+n}
```

where:

colnotation

Specifies a column (see *Selecting Report Columns* on page 11-33). Note that the column must be defined for TYPE=REPORT, and that it cannot be an ACROSSCOLUMN.

n

Is the amount of blank space, in UNITS, between the left margin and the beginning of the column.

This type of placement is called absolute positioning. It is easy to overlap columns using this method of column placement. For example, if you position a column at 1 inch from the margin and another column at 1.1 inch from the margin, the two columns will overlap if the first column is wider than 0.1 inch.

+*n*

Is the amount of blank space, in UNITS, between the end of the previous column and the beginning of this column.

This type of placement is called relative positioning.

Note: You can also specify column positions in a report request via spot markers and IN syntax; in this case the integer value in a column position refers to the number of spaces (measured in 12-point COURIER font) from the left side of the page. However, the StyleSheet POSITION attribute is the recommended method for positioning columns and heading elements.

Headings and footings have an additional placement option. You can align them with a particular column. The syntax is

```
TYPE = headtype[element,] POSITION = {colnotation|[+]n}
```

where:

headtype

Can be one of the following: TABHEADING, TABFOOTING, HEADING, FOOTING, SUBHEAD or SUBFOOT.

element

Specifies a subcomponent of the heading or footing (see *Selecting Headings and Footings* on page 11-26).

colnotation

Specifies a column (see *Selecting Report Columns* on page 11-33). The heading or footing element is placed where the specified column starts. FIELD elements are justified according to the justification of the column; TEXT elements are aligned with the left edge of the column.

n

Is the amount of blank space, in UNITS, between the left margin and the beginning of the column.

+*n*

Is the amount of blank space, in UNITS, between the previously printed item and the beginning of this element or component.

Determining Column Widths

The SQUEEZE attribute assigns column widths. See *Styling the Page Layout* on page 11-12 for a description.

Changing Column Sequence

The SEQUENCE attribute defines the order in which columns are displayed on a report.

By default, FOCUS prints BY fields first, then verb object fields in the order in which the request specifies them. For multi-verb requests, FOCUS prints the BY fields for the first verb, followed by the verb object fields of the first verb, followed by the remaining BY fields for the second verb, and the verb object fields for the second verb. For example:

```
TABLE FILE EMPLOYEE
SUM ED_HRS CURR_SAL
BY EMP_ID
PRINT ED_HRS CURR_SAL
BY EMP_ID
BY LAST_NAME
BY FIRST_NAME
END
```

The order of fields in the resulting report is EMP_ID, sum of ED_HRS, sum of CURR_SAL, LAST_NAME, FIRST_NAME, ED_HRS, CURR_SAL.

To change the sequence of a column in the report, select the column and define its sequence number. The syntax is

```
TYPE = REPORT, COLUMN = colnotation, SEQUENCE = nnn
```

where:

colnotation

Specifies a column (see *Selecting Report Columns* on page 11-33).

nnn

Is a positive integer that represents the column’s order in the report.

Note:

- SEQUENCE is valid only with TYPE=REPORT and COLUMN=colnotation. It reorders all information associated with a column: title, data, subtotal, and grandtotal.
- SEQUENCE is not valid with ACROSSCOLUMN or with any report output sorted across the page.
- SEQUENCE is not valid with fields that are displayed over each other using the OVER parameter. This includes the field immediately before the first OVER parameter.
- The sequence numbers you assign need not be in sequential order or in increments of one. FOCUS arranges the columns in lowest to highest order of the numbers you assign.

FOCUS assigns a number to each column in a report, starting with one on the far left. In the following request, the default column order is LAST_NAME, FIRST_NAME, EMP_ID, DEPARTMENT:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND DEPARTMENT
BY LAST_NAME
BY FIRST_NAME
END
```

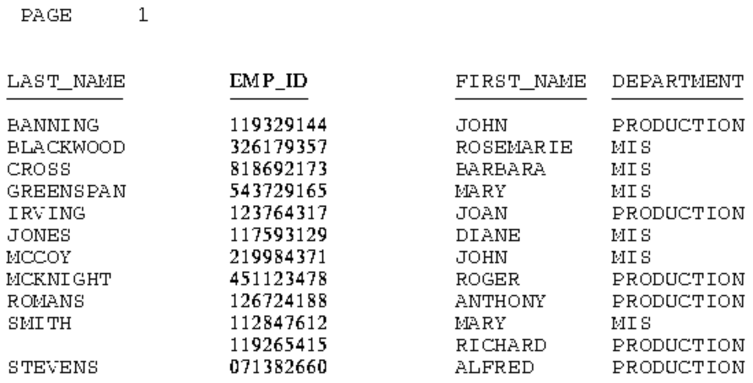
When activated for the request, the StyleSheet named SEQU places EMP_ID in the second column. The revised request and the SEQU StyleSheet follow:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND DEPARTMENT
BY LAST_NAME
BY FIRST_NAME
ON TABLE SET STYLE SEQU
ON TABLE HOLD AS SEQU FORMAT PS
END
```

StyleSheet SEQU:

```
TYPE=REPORT, COLUMN=EMP_ID, SEQUENCE=2, FONT=TIMES, $
```

In the resulting stylized report output, FOCUS shifts columns two and three (FIRST_NAME and DEPARTMENT) one column to the right so it can place EMP_ID in column two:



```
 PAGE     1

LAST_NAME        EMP_ID           FIRST_NAME  DEPARTMENT
---------        ------           ----------  ----------

BANNING          119329144        JOHN        PRODUCTION
BLACKWOOD        326179357        ROSEMARIE   MIS
CROSS            818692173        BARBARA     MIS
GREENSPAN        543729165        MARY        MIS
IRVING           123764317        JOAN        PRODUCTION
JONES            117593129        DIANE       MIS
MCCOY            219984371        JOHN        MIS
MCKNIGHT         451123478        ROGER       PRODUCTION
ROMANS           126724188        ANTHONY     PRODUCTION
SMITH            112847612        MARY        MIS
                 119265415        RICHARD     PRODUCTION
STEVENS          071382660        ALFRED      PRODUCTION
```

Consider the following StyleSheet:

```
TYPE=REPORT, COLUMN=DEPARTMENT, SEQUENCE=999, FONT=TIMES, $
```

With this StyleSheet and any request that prints the field DEPARTMENT, DEPARTMENT is always the last column in the stylized report output. To ensure that a column appears last in the report output, select a sequence number that is greater than the total number of columns.

Specifying Column Spacing

The LEFTGAP and RIGHTGAP attributes define how much space to leave between columns. The syntax is

```
TYPE=REPORT,[COLUMN=colnatation,] {RIGHTGAP|LEFTGAP} = n
```

where:

colnotation

Specifies a column (see *Selecting Report Columns* on page 11-33).

n

Is the amount of blank space, in UNITS, to the right or left of the column area.

LEFTGAP and RIGHTGAP are only valid with TYPE=REPORT and, optionally, with the qualifier COLUMN.

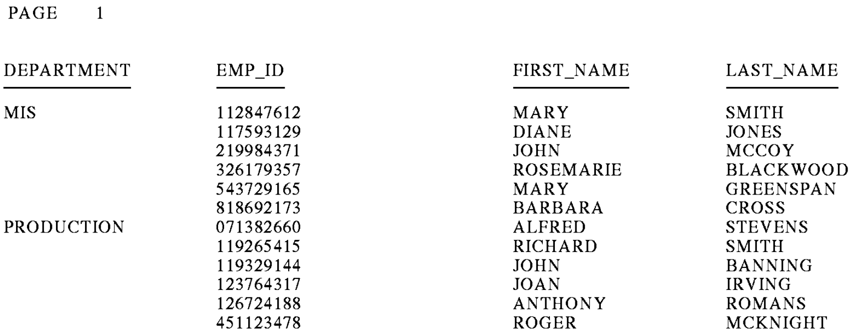
Consider the following TABLE request:

```
SET STYLE = SPACING
TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME
BY DEPARTMENT
ON TABLE HOLD AS SPACING FORMAT PS
END
```

StyleSheet SPACING follows:

```
TYPE=REPORT, FONT=TIMES, SIZE=10 ,$
TYPE=REPORT, COLUMN=EMP_ID, RIGHTGAP=1 ,$
```

In the resulting report, the RIGHTGAP attribute places one additional inch of space to the right of the EMP_ID column:

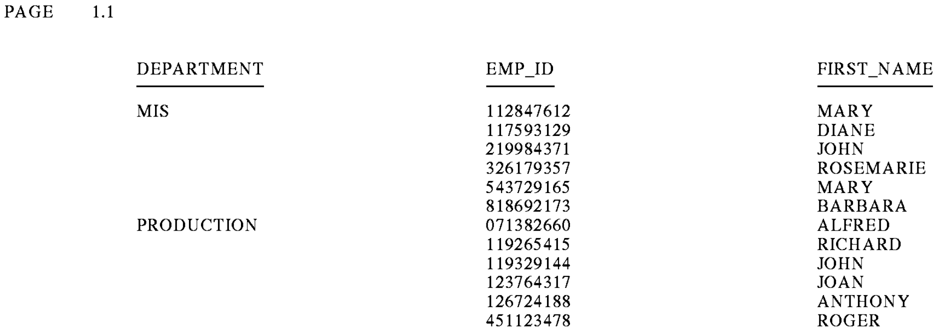


```
PAGE     1

DEPARTMENT        EMP_ID                   FIRST_NAME       LAST_NAME

MIS               112847612                MARY             SMITH
                  117593129                DIANE            JONES
                  219984371                JOHN             MCCOY
                  326179357                ROSEMARIE        BLACKWOOD
                  543729165                MARY             GREENSPAN
                  818692173                BARBARA          CROSS
PRODUCTION        071382660                ALFRED           STEVENS
                  119265415                RICHARD          SMITH
                  119329144                JOHN             BANNING
                  123764317                JOAN             IRVING
                  126724188                ANTHONY          ROMANS
                  451123478                ROGER            MCKNIGHT
```

The following request and StyleSheet place one inch before *every* column:

```
SET STYLE=RPTSPAC
TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME
BY DEPARTMENT
ON TABLE HOLD AS RPTSPAC FORMAT PS
END
TYPE=REPORT, FONT=TIMES, SIZE=10, LEFTGAP=1,$
```

Notice that the first column, DEPARTMENT, begins one inch from the left margin:



```
PAGE     1.1

           DEPARTMENT                 EMP_ID                    FIRST_NAME

           MIS                        112847612                 MARY
                                      117593129                 DIANE
                                      219984371                 JOHN
                                      326179357                 ROSEMARIE
                                      543729165                 MARY
                                      818692173                 BARBARA
           PRODUCTION                 071382660                 ALFRED
                                      119265415                 RICHARD
                                      119329144                 JOHN
                                      123764317                 JOAN
                                      126724188                 ANTHONY
                                      451123478                 ROGER
```

Interpreting Inheritance

System defaults define the style of a report component until you create a style that changes one or more of its characteristics. For example, the font for every report component is COURIER until you define a style that changes that font.

If a style listed in the StyleSheet lacks one or more style attributes (FONT, SIZE, STYLE, and COLOR), the components formatted by that style inherit values for the omitted attributes.

Report components automatically inherit characteristics from larger components as follows:

- The entire report inherits its characteristics from the default style.
- All report components inherit their characteristics from the values defined for the entire report.
- Elements in a report component inherit their characteristics from the component to which they belong. For example, a particular line in a multi-line heading inherits the characteristics of that heading.

Example Inheriting Styles

Consider the following example:

```
SET ORIENTATION = LANDSCAPE
SET SQUEEZE = ON

TABLE FILE EMPLOYEE
SUM AVE.GROSS AS 'AVERAGE,GROSS' AND GROSS AS 'GROSS,AMOUNT' AND
COMPUTE NET_PAY/D12.2 = GROSS - DED_AMT; AS 'NET,PAY'
BY HIGHEST PAY_DATE AS 'PAY DATE'
ACROSS DEPARTMENT
HEADING
"Swifty Information Group"
"Department Payroll Report"
" "
"Sorted down by Pay Date, (most recent first)"
"and across the page by Department"
" "
FOOTING
"Confidential Information "
IF DEPARTMENT IS MIS OR PRODUCTION
ON TABLE SET STYLE ACROSS
ON TABLE HOLD AS ACROSS FORMAT PS
END
```

StyleSheet ACROSS:

```
    TYPE=REPORT, FONT=TIMES, SIZE=10,$
    TYPE=HEADING, SIZE=16, STYLE=BOLD,$
    TYPE=HEADING, LINE=4, SIZE=14, STYLE=ITALIC,$
    TYPE=HEADING, LINE=5, SIZE=14, STYLE=ITALIC,$
1.  TYPE=ACROSSTITLE, SIZE=12, STYLE=UNDERLINE,$
2.  TYPE=ACROSSVALUE, STYLE=ITALIC, FONT=HELVETICA,$
3.  TYPE=TITLE, FONT=HELVETICA, SIZE=8,$
4.  TYPE=DATA, ACROSSCOLUMN=4, FONT=HELVETICA, SIZE=8,$
```

In the resulting report:

1. The ACROSSTITLE is DEPARTMENT, generated by the phrase ACROSS DEPARTMENT in the report request.

   The declaration for TYPE=ACROSSTITLE makes its size 12 points and its text style UNDERLINE. It inherits its font (TIMES) from TYPE=REPORT and its color (BLACK) from system defaults.

2. The ACROSS values are MIS and PRODUCTION, generated by the phrase ACROSS DEPARTMENT in the report request.

   The declaration for TYPE=ACROSSVALUE in the StyleSheet formats their font as HELVETICA and their text style as ITALIC. They inherit their size (10 points) from TYPE=REPORT and their color (BLACK) from system defaults.

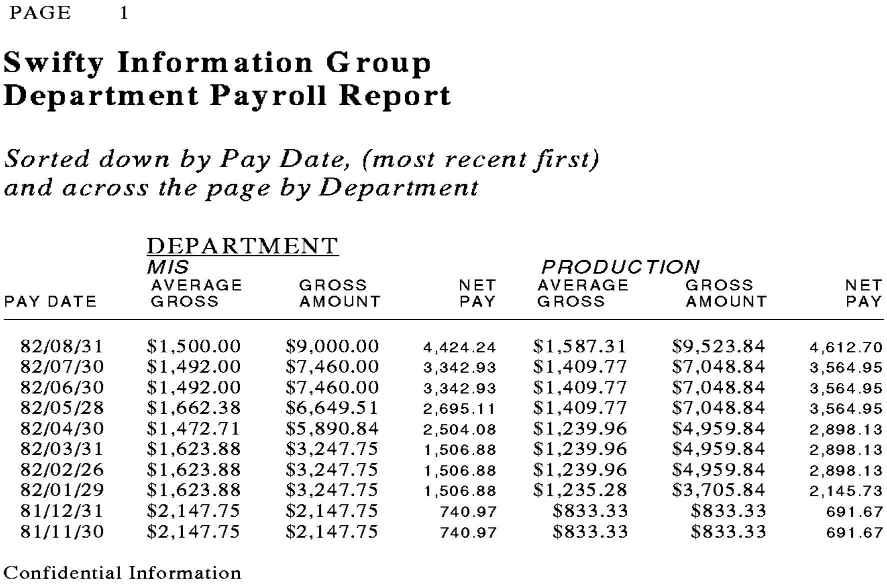
3. The column titles under each ACROSSVALUE are ‘AVERAGE GROSS’, ‘GROSS,AMOUNT’, and ‘NET,PAY’. There is one set of these titles under MIS and one set under PRODUCTION in the report. However, since the COMPUTE expression references DED_AMT, FOCUS adds DED_AMT as a NOPRINT field in each group between GROSS and NET_PAY. Therefore, NET_PAY becomes the fourth field.

   The declaration for TYPE=TITLE makes the font for column titles HELVETICA and their size 8 points. They inherit their text style (NORMAL) and their color (BLACK) from system defaults.

4. The declaration for TYPE=DATA, ACROSSCOLUMN=4, refers to the data values in the fourth column (the NET_PAY field; see item 3) under each ACROSSVALUE. This declaration makes their font HELVETICA and their size 8 points. They inherit their text style (NORMAL) and their color (BLACK) from system defaults.

The FOOTING inherits its style from TYPE=REPORT.

The report output (sized to fit) follows:



```
PAGE      1

Swifty Information Group
Department Payroll Report

Sorted down by Pay Date, (most recent first)
and across the page by Department

             DEPARTMENT
             MIS                                          PRODUCTION
             AVERAGE        GROSS           NET   AVERAGE        GROSS            NET
PAY DATE     GROSS          AMOUNT          PAY   GROSS          AMOUNT           PAY
------------------------------------------------------------------------------------

82/08/31     $1,500.00      $9,000.00  4,424.24   $1,587.31      $9,523.84   4,612.70
82/07/30     $1,492.00      $7,460.00  3,342.93   $1,409.77      $7,048.84   3,564.95
82/06/30     $1,492.00      $7,460.00  3,342.93   $1,409.77      $7,048.84   3,564.95
82/05/28     $1,662.38      $6,649.51  2,695.11   $1,409.77      $7,048.84   3,564.95
82/04/30     $1,472.71      $5,890.84  2,504.08   $1,239.96      $4,959.84   2,898.13
82/03/31     $1,623.88      $3,247.75  1,506.88   $1,239.96      $4,959.84   2,898.13
82/02/26     $1,623.88      $3,247.75  1,506.88   $1,239.96      $4,959.84   2,898.13
82/01/29     $1,623.88      $3,247.75  1,506.88   $1,235.28      $3,705.84   2,145.73
81/12/31     $2,147.75      $2,147.75    740.97     $833.33        $833.33     691.67
81/11/30     $2,147.75      $2,147.75    740.97     $833.33        $833.33     691.67

Confidential Information
```

Example Styling Subtotals and Grand Totals

The next example illustrates subtotals and grand total:

```
TABLE FILE EMPLOYEE
PRINT DED_AMT AS 'DEDUCTION'
BY EMP_ID AS 'EMPLOYEE'
BY LAST_NAME NOPRINT
BY FIRST_NAME NOPRINT
BY PAY_DATE AS 'PAY DATE'
BY DED_CODE AS 'DEDUCTION CODE'
ON EMP_ID SUBTOTAL
ON PAY_DATE SUBTOTAL
ON FIRST_NAME SUBHEAD
"<FIRST_NAME <HIRE_DATE "
HEADING
"EMPLOYEES WITH JOBCODE <CURR_JOBCODE "
" "
ON TABLE COLUMN-TOTAL
WHERE CURR_JOBCODE EQ 'B04';
WHERE PAY_DATE EQ 820730;
WHERE DED_CODE EQ 'HLTH' OR 'LIFE';
ON TABLE SET STYLE TOTALS
ON TABLE HOLD AS TOTALS FORMAT PS
END
```

StyleSheet TOTALS:

```
   TYPE=REPORT, FONT=TIMES,$
1. TYPE=SUBTOTAL, BY=B1, STYLE=BOLD,$
2. TYPE=SUBTOTAL, BY=B4, STYLE=ITALIC,$
3. TYPE=GRANDTOTAL, SIZE=14,$
   TYPE=DATA, COLUMN=B1, FONT=TIMES, STYLE=BOLD,$
   TYPE=DATA, COLUMN=B4, FONT=TIMES, STYLE=BOLD,$
```

In the resulting report:

1. The first BY column is EMPLOYEE, generated by the phrase BY EMP_ID AS 'EMPLOYEE' in the request.

   The declaration for TYPE=SUBTOTAL, BY=B1 makes the text style for the subtotal line generated by the phrase ON EMP_ID SUBTOTAL bold. This subtotal line inherits its font (TIMES) from TYPE=REPORT and its size and color (12 points and BLACK) from system defaults.

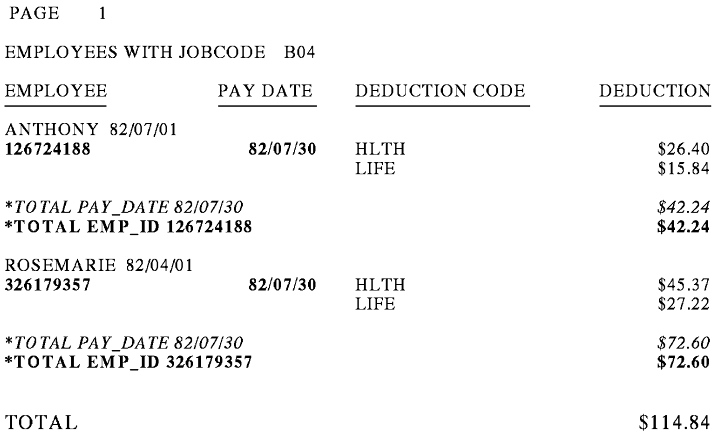
2. The fourth BY column is PAY DATE. It appears to be the second BY column because LAST_NAME and FIRST_NAME are NOPRINT fields.

   The declaration for TYPE=SUBTOTAL, BY=B4 makes the text style for the subtotal line generated by the phrase ON PAY_DATE SUBTOTAL italic. This subtotal line inherits its font (TIMES) from TYPE=REPORT and its size and color (12 points and BLACK) from system defaults.

3. The grand total is the last total in a report. In this case it is the total generated by the phrase ON TABLE COLUMN-TOTAL.

   The declaration for TYPE=GRANDTOTAL makes the size of this total 14 points. It inherits its font (TIMES) from TYPE=REPORT and its style and color (NORMAL and BLACK) from system defaults.

Following is the report output:



```
PAGE     1

EMPLOYEES WITH JOBCODE  B04

EMPLOYEE              PAY DATE    DEDUCTION CODE        DEDUCTION

ANTHONY  82/07/01
126724188             82/07/30    HLTH                     $26.40
                                  LIFE                     $15.84

*TOTAL PAY_DATE 82/07/30                                   $42.24
*TOTAL EMP_ID 126724188                                    $42.24

ROSEMARIE  82/04/01
326179357             82/07/30    HLTH                     $45.37
                                  LIFE                     $27.22

*TOTAL PAY_DATE 82/07/30                                   $72.60
*TOTAL EMP_ID 326179357                                    $72.60


TOTAL                                                     $114.84
```

Example Styling RECAP Lines

The final example illustrates RECAP lines:

```
TABLE FILE EMPLOYEE
SUM GROSS AS 'Gross Amount' AND DED_AMT AS 'Deduction Amt' AND
COMPUTE DG_RATIO/F4.2 = DED_AMT / GROSS; AS 'RATIO'
BY DEPARTMENT AS 'Department'
BY PAY_DATE AS 'Pay Date'
WHERE PAY_DATE EQ 820730
ON DEPARTMENT RECAP DEPT_NET/D8.2M=GROSS - DED_AMT;
ON PAY_DATE RECAP NET/D8.2M=GROSS - DED_AMT;
HEADING
" Pay Summary"
ON TABLE SET STYLE RECAP
ON TABLE HOLD AS RECAP FORMAT PS
END
```

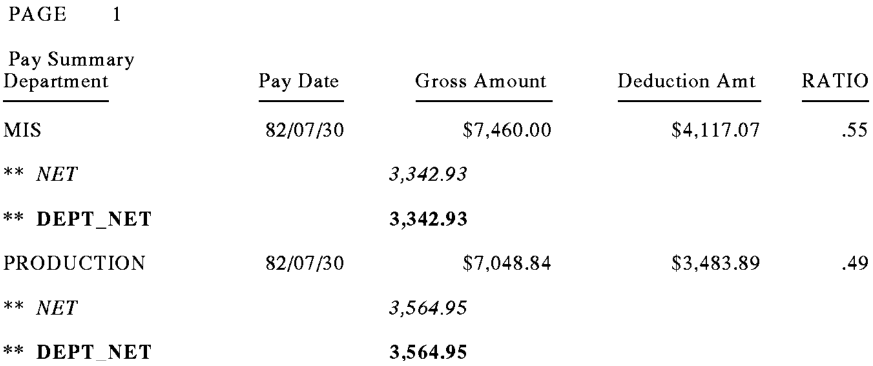
StyleSheet RECAP.

```
   TYPE=REPORT, FONT=TIMES, SIZE=10,$
1. TYPE=RECAP, BY=B2, STYLE=ITALIC,$
2. TYPE=RECAP, BY=B1, STYLE=BOLD,$
```

In the report:

1. The second BY field in the request is PAY_DATE. Therefore, the declaration for TYPE=RECAP, BY=B2 refers to the line in the report generated by the phrase ON PAY_DATE RECAP NET ... in the request. This declaration makes the NET line italic. It inherits its font and size (TIMES and 10 points) from TYPE=REPORT and its color (BLACK) from system defaults.

2. The first BY field is DEPARTMENT. Therefore, the declaration for TYPE=RECAP, BY=B1 refers to the line generated by the phrase ON DEPARTMENT RECAP DEPT_NET ... in the request. This declaration makes the DEPT_NET line bold. It inherits its font and size (TIMES and 10 points) from TYPE=REPORT and its color (BLACK) from system defaults.



```
 PAGE     1

 Pay Summary
Department              Pay Date     Gross Amount     Deduction Amt    RATIO

MIS                     82/07/30        $7,460.00         $4,117.07      .55

** NET                            3,342.93

** DEPT_NET                       3,342.93

PRODUCTION              82/07/30        $7,048.84         $3,483.89      .49

** NET                            3,564.95

** DEPT_NET                       3,564.95
```

Conditional Styling

Conditional styling, or stoplighting, formats a report component based on the results of a conditional test. The WHEN attribute specifies the test condition. The syntax is

```
WHEN = column  operator {value|column}
```

where:

column

Specifies a single column in the report (see *Selecting Report Columns* on page 11-33). You can include prefixes such as TOT. or MIN. in front of a field name as long as the prefixed fields appear in the TABLE request.

operator

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

value

Is a number or a string enclosed in single quotation marks. If it is a string, its case (for example, uppercase) must match the case of the data in the data source.

Note:

- FOCUS ignores WHEN conditions that refer to a column that is not referenced in the report request. If you want to base a WHEN condition on a field that you do not want to print in the report, make the field a NOPRINT field. For example:

  ```
  TABLE FILE CAR
  SUM MAX.SALES NOPRINT
  SUM SALES BY COUNTRY BY CAR
  END
  TYPE=HEADING, STYLE=BOLD, WHEN= MAX.SALES LT 10000 ,$
  ```

- Arithmetical and logical expressions are not allowed on either side of the operator. However, you can create a temporary field that evaluates the expression and use that temporary field in the WHEN expression. For example:

  ```
  DEFINE FILE CAR
  TEST/I2 = RETAIL_COST GT (1.1 * DEALER_COST);
  END
  TABLE FILE CAR
     .
     .
     .
  END
  TYPE=DATA, COLUMN=RETAIL_COST , COLOR=RED, WHEN=TEST NE 0 ,$
  ```

  The field TEST must be a field in the report request. If you do not want to print it in the report, make it a NOPRINT field.

- If more than one WHEN condition applies to a report component, FOCUS evaluates them in the order in which they appear in the StyleSheet. For example:

  ```
  TYPE=DATA, COLOR=RED, WHEN=RETAIL_COST GT 12000, $
  TYPE=DATA, COLOR=GREEN, WHEN=RETAIL_COST GT 8000,$
  ```

  Any data line in which the value of RETAIL_COST satisfies both conditions is printed in red because the WHEN condition for red comes first in the StyleSheet. You can use this feature to implement simple if-then-else logic. You can implement more complex logic by combining this technique with the use of temporary fields.

Example Conditional Styling

The following StyleSheet prints lines in red and bold when the value of the field SALES is greater than $1,000:

```
TYPE=DATA, STYLE=BOLD, COLOR=RED, WHEN=SALES GT 1000 ,$
```

The next StyleSheet prints the value in the column SALES in red if SALES is less than $100,000, or in green if SALES exceeds $200,000:

```
TYPE=DATA, COLUMN=SALES, COLOR=RED,   WHEN=SALES LT 100000 ,$
TYPE=DATA, COLUMN=SALES, COLOR=GREEN, WHEN=SALES GT 200000 ,$
```

This StyleSheet prints the data value in column C1 in bold whenever the value of SALES is greater than $1,000:

```
TYPE=DATA, COLUMN=C1, STYLE=BOLD, WHEN=SALES GT 1000 ,$
```

In the next example, all lines representing instances in which COUNTRY has the value ENGLAND are shown in bold. If COUNTRY is a sort (BY) field, the entire sort group is shown in bold, even though the value ENGLAND appears on only one line:

```
TYPE=DATA, WHEN=COUNTRY EQ 'ENGLAND', STYLE=BOLD ,$
```

The next example illustrates that for TYPE=REPORT, an entire report column—including the title, data, subtotal, and grand total—can change if the WHEN condition is based on an aggregate value:

```
TYPE=REPORT, COLUMN=C1, WHEN=TOT.SALES LT 100000, COLOR=RED,$
```

Using WHEN With ACROSSCOLUMN

If you use WHEN with ACROSSCOLUMN, styles are applied differently depending on whether the column referenced in the WHEN condition falls within ACROSS groups. A WHEN column that is within an ACROSS group controls the formatting of all data within the same ACROSS group.

Consider the following report output:

```
           SEATS
           4                                 5
COUNTRY    RETAIL_COST        DEALER_COST    RETAIL_COST    DEALER_COST
----------------------------------------------------------------------------------
ENGLAND    8000               7000           7500           6000
JAPAN      9000               8000           10000          9000
```

In this report, both RETAIL_COST and DEALER_COST are printed ACROSS SEATS. However, COUNTRY does not fall within the ACROSS group.

In the following StyleSheet, data values in the RETAIL_COST columns are formatted according to the data in their corresponding DEALER_COST columns:

```
TYPE=DATA, ACROSSCOLUMN=RETAIL_COST, COLOR=RED, WHEN=DEALER_COST GT 7100,$
```

Therefore, the value 9000 in the RETAIL_COST column under SEATS=4 is printed in red, since its corresponding DEALER_COST value (8000) is greater than 7100; the RETAIL_COST value 7500 is not printed in red because its corresponding DEALER_COST is not greater than 7100.

In the next StyleSheet, the WHEN condition references the column COUNTRY, which is not part of an ACROSS group:

```
TYPE=DATA, ACROSSCOLUMN=RETAIL_COST, COLOR=RED, WHEN=COUNTRY EQ 'ENGLAND',$
```

In this case all RETAIL_COST values in the line for ENGLAND are printed in red.

If a StyleSheet uses ACROSSCOLUMN with WHEN and a field name referenced in the WHEN condition appears both under the ACROSS and elsewhere in the report (as is possible with a multi-verb request), the field name under the ACROSS takes precedence. You can refer to the other column using another version of the column notation, such as Cn.

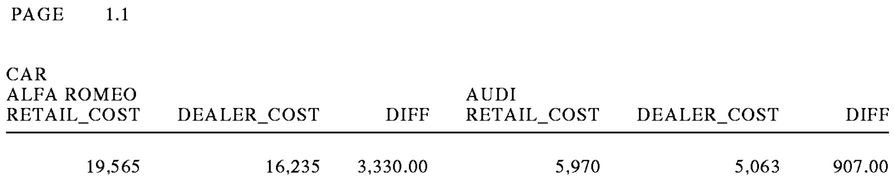
For example, in the next request, the RETAIL_COST column under each value of CAR may be printed in bold depending on the corresponding value of DIFF for each CAR:

```
TABLE FILE CAR
SUM RETAIL_COST AND DEALER_COST
AND COMPUTE DIFF/D12.2=RETAIL_COST - DEALER_COST;
ACROSS CAR
ON TABLE SET SQUEEZE ON
ON TABLE SET STYLE 120193A
ON TABLE HOLD AS 120193A FORMAT PS
END
```

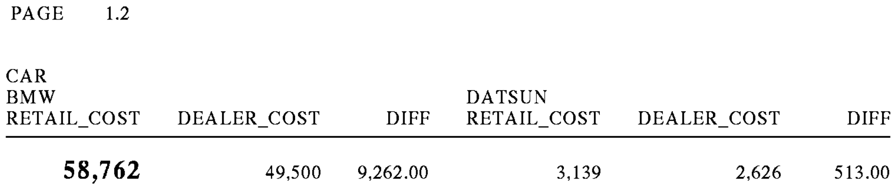
StyleSheet 120193A:

```
TYPE=REPORT, FONT=TIMES, SIZE=10,$
TYPE=DATA, ACROSSCOLUMN=RETAIL_COST, SIZE=14, WHEN=DIFF GT 9000, STYLE=BOLD,$
```

Two pages of the resulting report follow:



```
 PAGE     1.1

CAR
ALFA ROMEO                                  AUDI
RETAIL_COST   DEALER_COST        DIFF      RETAIL_COST   DEALER_COST        DIFF
------------------------------------------------------------------------------------

       19,565          16,235   3,330.00          5,970          5,063     907.00
```



```
PAGE     1.2

CAR
BMW                                         DATSUN
RETAIL_COST   DEALER_COST        DIFF      RETAIL_COST   DEALER_COST        DIFF
------------------------------------------------------------------------------------

       58,762          49,500   9,262.00          3,139          2,626     513.00
```

To specify the DIFF field outside the ACROSS, you can use the notation C3:

```
WHEN=C3 GT 9000
```

Example Conditional Styling With a BY Field

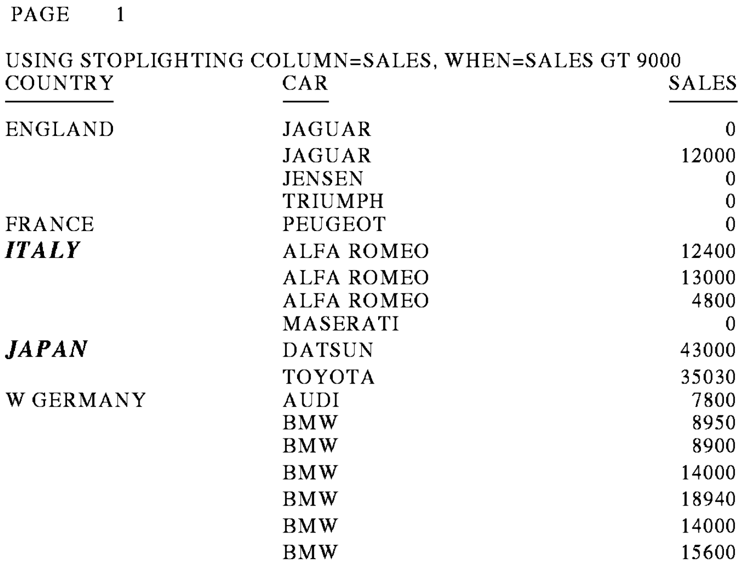
The following example uses the notation COLUMN = B1 to select the COUNTRY column and to make it bold, italic, and l4-point for lines with SALES greater than 9000:

```
TABLE FILE CAR
PRINT CAR AND SALES
BY COUNTRY
HEADING
"USING STOPLIGHTING COLUMN=SALES, WHEN=SALES GT 9000"
ON TABLE SET STYLE STOPLIT4
ON TABLE HOLD AS STOPLIT4 FORMAT PS
END
```

StyleSheet STOPLIT4:

```
TYPE=REPORT, FONT=TIMES,$
TYPE=DATA, COLUMN=B1, SIZE=14, STYLE=BOLD+ITALIC, WHEN=SALES GT 9000,$
```

The report output follows:



```
 PAGE      1

USING STOPLIGHTING COLUMN=SALES, WHEN=SALES GT 9000
COUNTRY               CAR                      SALES
-------               ---                      -----

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

Example Conditional Styling With A NOPRINT Field

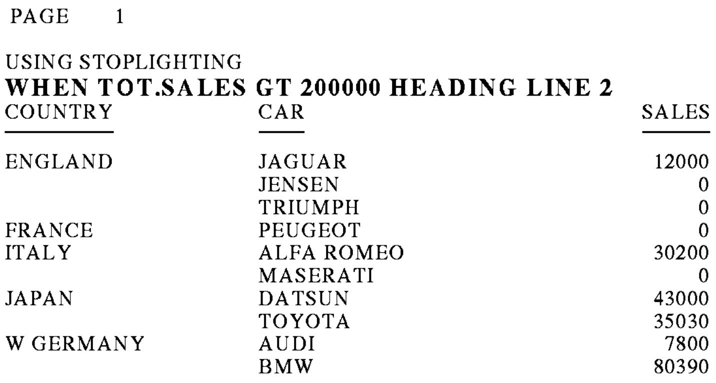
The next example uses the value of total sales in the WHEN condition. The column TOT.SALES must be referenced in the request or FOCUS ignores the WHEN condition. However, it is a NOPRINT field and is not printed in the report.

```
TABLE FILE CAR
SUM TOT.SALES NOPRINT
SUM SALES
BY COUNTRY
BY CAR
HEADING
"USING STOPLIGHTING"
"WHEN TOT.SALES GT 200000 HEADING LINE 2"
ON TABLE SET STYLE STOPLIT5
ON TABLE HOLD AS STOPLIT5 FORMAT PS
END
```

StyleSheet STOPLIT5:

```
TYPE=REPORT, FONT=TIMES, SIZE=10,$
TYPE=HEADING, LINE=2, SIZE=12, STYLE=BOLD,
    WHEN=TOT.SALES GT 200000,$
```

The report output follows:



```
 PAGE     1

USING STOPLIGHTING
WHEN TOT.SALES GT 200000 HEADING LINE 2
COUNTRY             CAR                          SALES
-------             ---                          -----

ENGLAND             JAGUAR                       12000
                    JENSEN                           0
                    TRIUMPH                          0
FRANCE              PEUGEOT                          0
ITALY               ALFA ROMEO                   30200
                    MASERATI                         0
JAPAN               DATSUN                       43000
                    TOYOTA                       35030
W GERMANY           AUDI                          7800
                    BMW                          80390
```

12 Saving and Reusing Your Report Output

Topics:

- Extract File Types
- The HOLD File
- Keyed Retrievals From FOCUS Extract Files
- The PCHOLD File: Downloading Files to a PC
- Controlling Attributes in HOLD Master Files
- The SAVE File
- The SAVB File

FOCUS provides options that enable you to save report data in data files, rather than display the report on the screen. You can save report output as a data file to create complex multi-step reports, to download information to another computer, or to convert data to another format such as a FOCUS database, a relational table, or a spreadsheet workfile. You can:

- Create a subset of the original data file.
- Generate two-step reports.
- Convert information in one type of data file to a FOCUS database or relational table.
- Format report output for use by other software, including spreadsheets and word processors.

Extract files remain usable until they are erased or written over. Subsequent extract files created during a session replace the initial versions. Hence, only one extract file of each type can be active at one time, unless you give it another name by using the AS phrase.

FOCUS automatically issues a FILEDEF or ALLOCATE command when creating extract files. The ddname used to identify the file is the same as the name of the extract file (HOLD, SAVE, or SAVB), if not already allocated.

You can preserve the internal matrix and keep it available for subsequent HOLD, SAVE, SAVB, and RETYPE commands by setting SAVEMATRIX to ON.

Extract File Types

There are four types of files, called extract files, in which output can be saved:

- **HOLD.** The HOLD option creates a data file containing the results of a report request. You can specify that the HOLD file's data be in internal format (the default), be formatted as a FOCUS database, be formatted as a simple character file, or be in a format suitable for use by a variety of other software products. For some formats, HOLD also creates a corresponding Master File. You can then write other report requests that in turn extract data from the HOLD file.
- **PCHOLD.** This is similar to HOLD, with the advantage that both the data file and the optional Master File are automatically downloaded to a client computer running FOCUS. As with a HOLD file, you can specify a variety of file formats.
- **SAVE.** This option is identical to a HOLD file, except that it does not create a Master File, and the default format is character, not internal. As with a HOLD file, you can specify a variety of formats suitable for use with other software products.
- **SAVB.** This option is identical to a HOLD file in internal format, except that it does not create a Master File.

The HOLD File

The FOCUS HOLD command enables you to retrieve and process data, then extract the results for further processing. That is, your TABLE request can create a new data file, complete with a corresponding Master File (depending on which data format you chose), from which you can generate new reports. If you create a HOLD file in FOCUS format, you can update it using the MODIFY, MAINTAIN, SCAN, and FSCAN facilities.

The HOLD file is a sequential data file. When created in internal FOCUS format, it is a single-segment data file. The HOLD Master File—if one is created—is a subset of the original Master File, and may also contain fields that have been created using COMPUTE or DEFINE or have been generated in an ACROSS.

When a HOLD file is created in internal format (the default), fields with format I remain four-byte binary integers; format F fields remain in four-byte floating-point format; format D fields remain in eight-byte double-precision floating-point; and format P fields remain in packed decimal notation and occupy eight bytes (for fields less than or equal to eight-bytes long) or 16 bytes (for packed fields longer than eight bytes). Alphanumeric fields (format A) are stored in character format.

Every data field in the sequential extract record is aligned on the start of a full 4-byte word. Therefore, if the format is A1, the field is padded with three bytes of blanks on the right. This alignment makes it easier for user-coded subroutines to process these data fields.

Sometimes, an application requires a data format that is not among the HOLD FORMAT options. The subroutine, invoked by the VIA option, processes each output record as it is written to the HOLD data file. For information on writing subroutines for use with HOLD files, see Technical Memo 7810, *Writing User-Coded Programs for HOLD Files*.

The output Master File contains only the fields in the report request. The fields in a HOLD file have the original names that would be retrieved had the report been displayed or printed. If you use the AS phrase to rename a field in your request, the AS name appears in the HOLD file when the SET ASNAMES=ON command is specified. Aliases, however, are not carried over into the HOLD Master. A new set of aliases is supplied automatically. These aliases are named E01 for the first field, E02 for the second, and so forth.

Syntax How to Create a HOLD Extract File

You can specify a HOLD file from a report request using the following syntax:

```
ON TABLE HOLD [AS filename][FORMAT fmt][MISSING {ON|OFF}][VIA program]
```

or

```
display_field HOLD [AS filename][FORMAT fmt][MISSING {ON|OFF}][VIA program]
```

You can also specify a HOLD file from the FOCUS command line or from within Hot Screen. When you issue HOLD from the FOCUS command line, do so after the report is displayed.

You can issue HOLD from the Hot Screen command line at any time while a report is displayed and on any page of the report. Regardless of the page from which you issue the command, the entire report is saved, and a data file and Master File are created for that report (just as they are when you issue the HOLD or PCHOLD command from within a TABLE request, or after exiting Hot Screen).

Note: You cannot use the SAVE or SAVB commands from within Hot Screen. You must include these commands in a TABLE request, or issue them from the FOCUS command level after exiting Hot Screen.

You can issue multiple HOLD commands for a single TABLE request; however, once you specify the FOCUS format with a HOLD command from within Hot Screen, you cannot issue another HOLD command during that Hot Screen session.

Syntax for issuing HOLD from the FOCUS and Hot Screen command lines is the same:

```
HOLD [AS filename][FORMAT fmt][MISSING {ON|OFF}][VIA program]
```

`AS fileame`

Specifies a file name for your HOLD file. If you do not specify a file name, it defaults to HOLD. Each HOLD command will overwrite the previous HOLD file. Coding a distinct file name in each request will direct the hold data to separate files.

`FORMAT fmt`

Specifies the format of the HOLD extract file.

`HTML` creates a complete HTML document that can be viewed in a Web browser. **Note:** Available as a feature of the Web Interface for FOCUS.

`HTMTABLE` creates an output file that contains only an HTML table. The output produced is not a complete HTML document. This file can be included in another HTML document using the FOCUS command -HTMLFORM. **Note:** Available as a feature of the Web Interface for FOCUS.

`ALPHA` creates a fixed-format file in character format. Text fields are supported.

`DIF` captures the report output and creates a comma-delimited file for use with most spreadsheet packages.

`EXCEL` captures the report output, including headings, in a binary file that can be transferred to another environment such as a Web server or a PC and opened in Microsoft Excel. The recommended transfer mechanism is FTP in binary mode. In CMS, the file type of the resulting file is XLS. On a PC, the extension should be .xls.

`LOTUS` captures all the columns of the FOCUS report request in a character file which LOTUS 1-2-3 can then import. All alphanumeric fields are enclosed in quotation marks. Columns are separated by commas. The maximum record length is 512 bytes. The file type in CMS is PRN.

`SYLK` captures all the columns of the report request in a character file for Microsoft's spreadsheet program Multiplan. The generated file cannot have more than 9,999 rows.

`DB2` creates a DB2 table. **Note:** Available as a feature of the DB2 Interface.

`FOCUS [INDEX fld1...]` creates a FOCUS database. See *HOLD Files in FOCUS Format* on page 12-11.

`ORACLE` creates an Oracle table. **Note:** Available as a feature of the Oracle Interface.

`SQL` creates an SQL/DS table. **Note:** Available as a feature of the SQL/DS Interface.

`SQLDBC` creates a Teradata table. **Note:** Available as a feature of the Teradata Interface.

`POSTSCRIPT` creates an output file in PostScript format. In CMS, the file type is PS.

`PS` is an abbreviation for POSTSCRIPT.

`PDF` creates an output file in PDF (Portable Document Format). **Note:** Available as a feature of the Web Interface for FOCUS. You will need the Adobe Acrobat Reader, available from Adobe Systems, Inc., to view files saved in this format.

WP captures the entire report output—including headings, footings, and subtotals—and creates a text file that can be easily incorporated into most word processing packages. To control whether a carriage control character is included in column 1 of each page of the report output use the following syntax [ON TABLE] HOLD AS *filename* FORMAT WP [CC|NOCC]. NOCC excludes carriage control characters. CC includes carriage control characters. In MVS, the HOLD file is created with RECFM VBA whenever carriage control is included. To include page control information in the file, you can also specify the TABPAGENO option in a heading or the SET PAGE=OFF command. A report saved in WP format uses the character 1 in column 1 to indicate page control information. Text fields are supported.

MISSING

Controls whether fields with the attribute MISSING=ON in the Master File are carried over into the HOLD file. MISSING ON is the default. If the HOLD command specifies MISSING OFF, the MISSING attribute is not carried over.

VIA *program*

For external applications only. It is a user-coded program that creates the HOLD data file. It defaults to the FOCUS internal format for fields; the ALPHA option is also available. Other formats, such as FOCUS or LOTUS, are not available when you use a program with the HOLD command.

Example Creating a HOLD Extract File

For example, the following request will create a HOLD file containing data generated by the request:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AND ED_HRS
BY DEPARTMENT
LIST CURR_SAL AND ED_HRS AND BANK_ACCT
BY DEPARTMENT
BY LAST_NAME BY FIRST_NAME
ON TABLE HOLD
END
```

FOCUS responds with the following message:

```
NUMBER OF RECORDS IN TABLE=   12 LINES=      12

HOLDING...
```

To display the report generated by this request, you would issue a TABLE request against the HOLD file or issue the RETYPE command.

Rules for Text Fields in HOLD and SAVE Files

Text fields can be used with HOLD files that have the following formats: ALPHA, WP, and FOCUS. They can also be used with SAVE and SAVB files. **Note:** Due to limitations to the use of text fields with FIXFORM (see the *Maintaining Databases* manual), the following restrictions apply when using text fields with the HOLD option.

- When you create a HOLD file, you can include as many text fields in the file as needed. However, you must specify text fields after non-text fields in the display list (PRINT…, SUM…, and so forth).
- When you create a HOLD file with a request that includes an ACROSS phrase, you can specify only one text field in the display list, and no non-text fields.
- When you use FIXFORM FROM HOLD, the HOLD file may not contain more than one text field and the text field must be the last field in the Master File.
- When HOLD files are read via FIXFORM, the interpretation of the missing text depends on whether the field's designation is MISSING=ON in the Master File, or conditional (C) in the FIXFORM format description, or some combination of the two.

For example,

```
TABLE FILE COURSES
PRINT COURSE_CODE DESCRIPTION
ON TABLE HOLD AS CRSEHOLD
END
```

will produce the following data in the HOLD file CRSEHOLD:

```
101 This course provides the DP professional with the skills
needed to create, maintain, and report from FOCUS databases.
%$
200 Anyone responsible for designing FOCUS databases will benefit
from this course, which provides the skills needed to design large,
complex databases and tune existing ones.
%$
201 This is a course in FOCUS efficiencies.
%$
```

The first record of the HOLD file contains data for COURSE_CODE 101, followed by the DESCRIPTION field. The data for this text field extends into the next record, beginning at Column 1, and continues to the end of the HOLD record. It is immediately followed by the end-of-text mark (%$) on a line by itself. The next record contains new data for the next COURSE_CODE and DESCRIPTION.

If the report uses two text fields, the first record will contain data for the first text field. After the end-of-text mark is written, the next text field will be displayed. This formatting applies to all file formats except WP, in which the report is saved exactly as it appears on the screen.

Missing Data for Text Fields in HOLD or SAVE Files

The following rules apply to missing data for text fields in HOLD and SAVE files:

- A blank line is valid data. An end-of-text mark will indicate the end of the field.
- If there is no text for a field, a single dot (.) followed by blanks will appear in the HOLD file.
- If MISSING=ON during the HOLD or SAVE process, a dot will be written out to the HOLD or SAVE file.
- If MISSING=OFF during the HOLD or SAVE process, a blank will be written out to the HOLD or SAVE file.

Displaying the HOLD Master

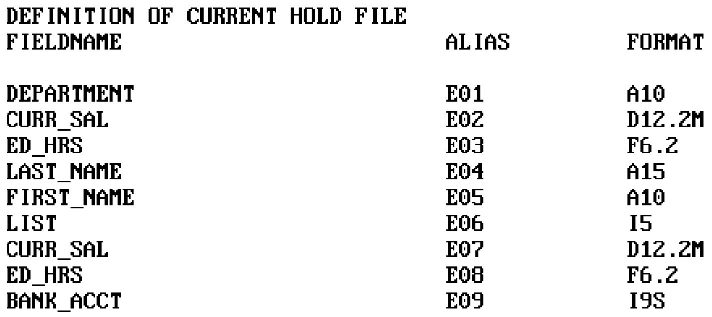
If the HOLD format option you selected created a Master File, you can display the fields, aliases, and formats from the Master with the command:

```
? HOLD
```

This command shows field names up to 32 characters. If a field name exceeds 32 characters, a caret(>) in the 32nd position indicates a longer field name.

If you have renamed the HOLD file using AS *filename*, use the following syntax to display the Master File of the HOLD file:

```
? HOLD filename
```



```
DEFINITION OF CURRENT HOLD FILE
FIELDNAME                        ALIAS         FORMAT

DEPARTMENT                       E01           A10
CURR_SAL                         E02           D12.2M
ED_HRS                           E03           F6.2
LAST_NAME                        E04           A15
FIRST_NAME                       E05           A10
LIST                             E06           I5
CURR_SAL                         E07           D12.2M
ED_HRS                           E08           F6.2
BANK_ACCT                        E09           I9S
```

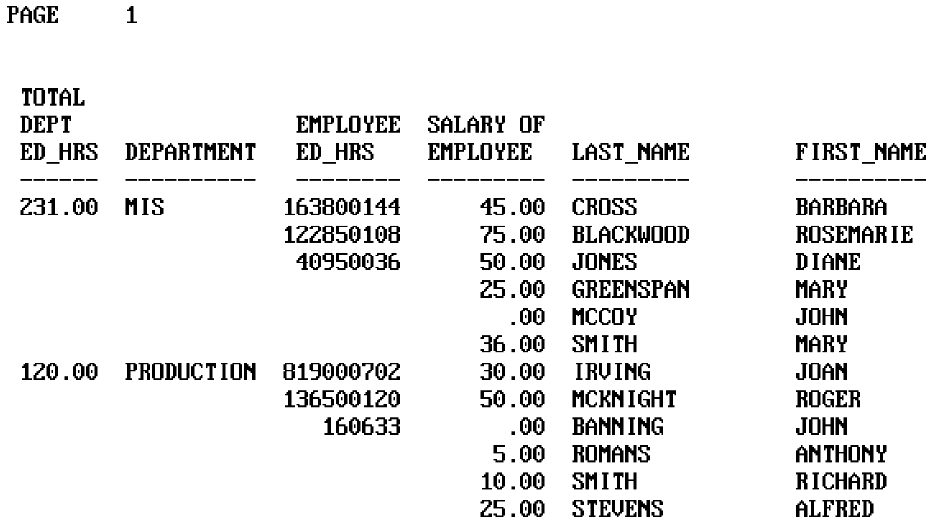
Notice that the formats shown are the values of the FORMAT attribute. You can see the values of the ACTUAL attribute by displaying the HOLD Master File using TED or any other editor. USAGE and ACTUAL formats for text fields specify only the length of the first line of each logical record in the HOLD file. The USAGE format is the same as the field format in the original Master File. The ACTUAL format is rounded up to a full (internal) word boundary, as is done for alphanumeric fields.

For information on how to control the TITLE, ACCEPT, and FIELD NAME attributes in a HOLD Master with the SET HOLDATTR and SET ASNAMES commands, see *Controlling Attributes in HOLD Master Files* on page 12-15.

Having created a HOLD file, you can now report from it. For example, this request against the HOLD file created previously

```
TABLE FILE HOLD
PRINT E07 AS 'SALARY OF,EMPLOYEE' AND LAST_NAME AND FIRST_NAME
BY HIGHEST E03 AS 'TOTAL,DEPT,ED_HRS'
BY E01
BY HIGHEST E08 AS 'EMPLOYEE,ED_HRS'
END
```

produces the following report:



```
PAGE     1


 TOTAL
 DEPT                   EMPLOYEE  SALARY OF
 ED_HRS  DEPARTMENT     ED_HRS    EMPLOYEE   LAST_NAME        FIRST_NAME
 ------  ----------     --------  ---------  ---------        ----------
 231.00  MIS           163800144      45.00  CROSS            BARBARA
                       122850108      75.00  BLACKWOOD        ROSEMARIE
                        40950036      50.00  JONES            DIANE
                                      25.00  GREENSPAN        MARY
                                        .00  MCCOY            JOHN
                                      36.00  SMITH            MARY
 120.00  PRODUCTION    819000702      30.00  IRVING           JOAN
                       136500120      50.00  MCKNIGHT         ROGER
                          160633        .00  BANNING          JOHN
                                       5.00  ROMANS           ANTHONY
                                      10.00  SMITH            RICHARD
                                      25.00  STEVENS          ALFRED
```

Keyed Retrievals From FOCUS Extract Files

Keyed retrieval is supported with any single segment SUFFIX=FIX file or FOCUS HOLD file. This can greatly reduce the IOs incurred in reading extract files. The performance gains are accomplished by using the SEGTYPE parameter in the Master File to identify which fields comprise the logical key for sequential files. With FIXRETRIEVE=ON, FOCUS stops the retrieval process when an equality or range test on the key holds true. In contrast, with FIXRETRIEVE=OFF, FOCUS reads all of the records from the sequential file and applies the screening conditions when creating the final report.

Syntax How to Control Keyed Retrieval for an Extract File

The syntax is

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

where ON is the default.

You can abbreviate this as:

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

Keyed Retrieval From an Extract File

The ON TABLE HOLD command allows you to read one of the many data sources FOCUS supports and create extract files. You can then join these fixed format sequential files to other data sources to narrow your view of the data. The concept of a logical key in a fixed format file permits qualified keyed searches for all records that match IF/WHERE tests for the first *n* KEY fields identified by the SEGTYPE attribute. Retrieval stops when the screening test detects values greater than those specified in the IF/WHERE test.

When FOCUS creates a Master File for the extract file, it adds a SEGTYPE of either S*n* or SH*n* based on the BY fields in the request. For example, PRINT *field* BY *field* creates a HOLD Master File with SEGTYPE=S1. Using BY HIGHEST *field* creates a Master with SEGTYPE=SH1.

Example Master File for Keyed Retrieval From an Extract File

The following Master File describes a fixed format sequential file with sorted values of MKEY in ascending order from 1 to 100,000:

```
FILE=SORTED,SUFFIX=FIX,$
SEGNAME=ONE,SEGTYPE=S1,$
 FIELD=MYKEY,MK,I8,I8,$
 FIELD=MFIELD,MF,A10,A10,$

TABLE FILE SORTED
 PRINT MFIELD
 WHERE MYKEY EQ 100
END
```

In this instance, with FIXRETRIEVE=ON, retrieval stops when MYKEY reaches 101, vastly reducing the potential number of IOs, as FOCUS reads only 101 records out of a possible 100K records.

Example Selection Criteria for Keyed Retrieval From an Extract File

Selection criteria may include lists of equality values. For example,

```
{IF|WHERE} MYKEY EQ x OR y OR z
```

IF/WHERE tests may also include range tests. For example,

```
{IF|WHERE} MYKEY IS-FROM x TO y
```

The maximum number of BY fields remains 32.

Keep in mind in using this feature that when adding unsorted records to a sorted HOLD file, out of sequence records will not be retrieved. For example, if a sorted file contains the following three records:

```
Key

1 1200

2 2340

3 4875
```

and you add the following record at the bottom of the file:

```
1 1620
```

With FIXRETRIEVE=ON, the new record with a key value of 1 would be omitted, as retrieval would stop as soon as a key value of 2 was encountered.

HOLD Files in FOCUS Format

The HOLD command creates a data file and a corresponding Master File using data extracted by TABLE requests. (The HOLD statement in TABLE requests also creates a data file and Master File.) HOLD can create files in FOCUS format. This feature gives you two capabilities:

- It enables you to create FOCUS files from any file that FOCUS can read.
- It enables you to create a subset of an existing FOCUS file.

Syntax How to Create HOLD Files in FOCUS Format

In a report request, the syntax is:

```
ON TABLE HOLD [AS filename] FORMAT FOCUS [INDEX field1 field2 ...]
```

After a report request or in Hot Screen, the syntax is

```
HOLD [AS filename] FORMAT FOCUS [INDEX field1 field2 ...]
```

where:

`AS` *`filename`*

Specifies a file name for your HOLD file. If you do not specify a file name, it defaults to HOLD. If you do not specify a file name, subsequent creations of HOLD files will overlay each other.

`INDEX` *`field1`*`...`

Enables you to index FOCUS fields.

Note: Once you use this format from within Hot Screen, you cannot issue another HOLD command while in the same Hot Screen session.

Reference Controlling the FOCUS File Structure

The structure of the FOCUS file being created varies according to the TABLE request. The rules are as follows:

- Each aggregation verb (SUM, COUNT, WRITE) creates a segment, with each new BY field in the request becoming a key. In a multi-verb request, the key to any created segment does not contain keys that are in the parent segment.
- If a PRINT or LIST verb is used to create a segment, all the BY fields, together with the internal FOCLIST field, form the key.
- All fields specified after the INDEX keyword will be indexed; that is, FIELDTYPE=I will be specified in the Master File. Up to four fields may be indexed.

In CMS a USE command is issued and the new database and Master File are created on whatever disk has WRITE permission and the most available space.

In MVS, the HOLD file is dynamically allocated if it is not currently allocated. This means the system may delete the file at the end of the session, even if you have not. Since HOLD files are usually deleted, this is a desired default; however, if you want to save the file, we recommend that it be allocated to ddname HOLDMAST as a permanent data set; the allocation can be performed within the standard FOCUS CLIST. Note that ddname HOLDMAST must not refer to the same PDS's referred to by the MASTER and FOCEXEC ddnames.

When FOCUS executes a HOLD FORMAT FOCUS statement, it creates a Master File for the HOLD file and a sequential data file called FOC$HOLD. It then loads the data in FOC$HOLD into the HOLD file using an internally generated MODIFY procedure.

To control whether the ACCEPT and TITLE attributes are propagated in the Master of the HOLD file, use the SET HOLDATTR command. To control the FIELDNAME attribute in the Master of the HOLD file, use the SET ASNAMES command.

For information on how to control the TITLE, ACCEPT, and FIELDNAME attributes in a HOLD Master File with the SET ASNAMES and SET HOLDATTR commands, see *Controlling Attributes in HOLD Master Files* on page 12-15.

Example Creating a HOLD File in FOCUS Format

The following example creates a subset of the CAR file:

```
TABLE FILE CAR
SUM SALES BY COUNTRY BY CAR BY MODEL
ON TABLE HOLD AS X1 FORMAT FOCUS
END
```

This request creates a single-segment FOCUS file with a SEGTYPE of S3 (because it has three BY fields) named X1 FOCUS.

The X1 Master File created by this request is:

```
FILE=X1                                     ,SUFFIX=FOC
SEGNAME=SEG01,        SEGTYPE=S03
FIELDNAME      =COUNTRY       ,E01    ,A10    ,$
FIELDNAME      =CAR           ,E02    ,A16    ,$
FIELDNAME      =MODEL         ,E03    ,A24    ,$
FIELDNAME      =SALES         ,E05    ,I6     ,$
```

Example Using PRINT to Create a FOCUS Database With a FOCLIST Field

This example creates a single-segment FOCUS file with a SEGTYPE of S4. Since this is a PRINT request, S4 stands for the addition of the three new BY fields and the FOCLIST value.

```
TABLE FILE CAR
PRINT SALES BY COUNTRY BY CAR BY MODEL
ON TABLE HOLD AS X2 FORMAT FOCUS INDEX MODEL
END
```

The Master File created by this request is:

```
FILE=X2                               ,SUFFIX=FOC
SEGNAME=SEG01,        SEGTYPE=S04
FIELDNAME      =COUNTRY       ,E01    ,A10    ,$
FIELDNAME      =CAR           ,E02    ,A16    ,$
FIELDNAME      =MODEL         ,E03    ,A24    ,FIELDTYPE=I,$
FIELDNAME      =FOCLIST       ,E02    ,I5     ,$
FIELDNAME      =SALES         ,E05    ,I6     ,$
```

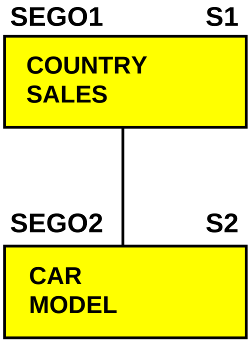
Example Creating a Two-Segment FOCUS Database

The following request contains two SUM verbs. The first, SUM SALES BY COUNTRY, creates a segment, with COUNTRY as the key and the summed values of SALES as a data field. The second, SUM SALES BY COUNTRY BY CAR BY MODEL, creates a descendant segment, with CAR and MODEL as the keys and SALES as a non-key field.

Note: The COUNTRY field does not form part of the key to the second segment. COUNTRY is a key in the path to the second segment; any repetition of this value will be redundant.

```
TABLE FILE CAR
SUM SALES BY COUNTRY
SUM SALES BY COUNTRY BY CAR BY MODEL
ON TABLE HOLD AS X3 FORMAT FOCUS
END
```

FOCUS creates a two-segment FOCUS file with the following structure:



The Master File for this newly-created FOCUS file is:

```
FILE=X3                              ,SUFFIX=FOC
SEGNAME=SEG01, SEGTYPE=S01
FIELDNAME             =COUNTRY       ,E01    ,A10    ,$
FIELDNAME             =SALES  ,E02   ,I6     ,$
SEGNAME=SEG02, SEGTYPE=S02,PARENT=SEG01
FIELDNAME             =CAR    ,E03   ,A16    ,$
FIELDNAME             =MODEL  ,E04   ,A24    ,$
FIELDNAME             =SALES  ,E05   ,I6     ,$
```

Example Creating a Three-Segment FOCUS Database

In this example, each verb creates one segment.

The key to the root segment is the new BY field, COUNTRY, while the keys to the descendant segments are the new BY fields. The last segment uses the internal FOCLIST field as part of the key, since the verb in that sentence is PRINT.

```
TABLE FILE CAR
SUM SALES BY COUNTRY BY CAR
SUM SALES BY COUNTRY BY CAR BY MODEL
PRINT SALES BY COUNTRY BY CAR BY MODEL BY BODY
ON TABLE HOLD AS X4 FORMAT FOCUS INDEX COUNTRY MODEL
END
```

The Master File is:

```
FILE=X4                                      ,SUFFIX=FOC
SEGNAME=SEG01, SEGTYPE =S02
FIELDNAME             =COUNTRY       ,E01    ,A10    ,FIELDTYPE=I,$
FIELDNAME             =CAR    ,E02   ,A16    ,$
FIELDNAME             =SALES  ,E03   ,I6     ,$
SEGNAME=SEG02, SEGTYPE =S01 ,PARENT=SEG01
FIELDNAME             =MODEL  ,E04   ,A24    ,FIELDTYPE=I,$
FIELDNAME             =SALES  ,E05   ,I6     ,$
SEGNAME=SEG03, SEGTYPE =S02 ,PARENT=SEG02
FIELDNAME             =BODYTYPE      ,E06    ,A12    ,$
FIELDNAME             =FOCLIST       ,E07    ,I5     ,$
FIELDNAME             =SALES  ,E08   ,I6     ,$
```

The PCHOLD File: Downloading Files to a PC

The PCHOLD command enables you to use data in a server environment to create HOLD files and automatically download those files to a client computer.

The syntax for PCHOLD in a TABLE request is

```
ON TABLE {PCHOLD|HOLD AT CLIENT} [AS filename] [FORMAT fmt]
```

where:

`AS filename`

Specifies a file name for your PCHOLD file. If you do not specify a file name, it defaults to PCHOLD. If you do not specify a file name, subsequent PCHOLD files will overlay each other.

`FORMAT fmt`

Specifies the format of the PCHOLD extract file.

Controlling Attributes in HOLD Master Files

The commands SET ASNAMES, SET HOLDLIST, and SET HOLDATTR enable you to control the FIELDNAME, TITLE, and ACCEPT attributes in HOLD Master Files.

- The SET ASNAMES command causes text specified in an AS phrase to be used as the field name in the HOLD Master File, and to be concatenated to the beginning of the first field name specified in an ACROSS phrase. SET ASNAMES is issued prior to the TABLE request and remains in effect for the duration of the session, unless it is changed during the session.
- The SET HOLDLIST command controls whether noprinted display fields are propagated to extract files. SET HOLDLIST is issued prior to or within a TABLE request, and remains in effect for the duration of the session, unless it is changed during the session.
- The SET HOLDATTR command causes TITLE and ACCEPT attributes used in the original Master File to be used in the HOLD Master File. SET HOLDATTR is issued prior to the TABLE request and remains in effect for the duration of the session, unless it is changed during the session.

The command SET HOLDSTAT enables you to include comments and DBA information in the HOLD Master File. For more information about SET HOLDSTAT, see the *Describing Data* manual.

Controlling Field Names in the HOLD Master File

When SET ASNAMES is set to ON or FOCUS, the literal specified in an AS phrase in a TABLE request is used as the field name in a HOLD Master File.

Syntax How to Control Field Names

The syntax is:

```
SET ASNAMES = value
```

Possible values are:

`ON`

Uses the literal specified in an AS phrase for the field name and controls the way ACROSS fields are named in HOLD files of any format.

`OFF`

Does not use the literal specified in an AS phrase as a field name in HOLD files, or affect the way ACROSS fields are named.

`FOCUS`

Uses the literal specified in an AS phrase as the field name and controls the way ACROSS fields are named only in HOLD files in FOCUS format. This is the default.

Reference Usage Notes for Controlling Field Names

If no AS phrase is specified for a field, FOCUS will use the field name from the original Master File. FOCUS will not use the TITLE attribute specified in the Master File unless SET HOLDATTR is used.

To ensure that fields referenced more than once in a request have unique field names in the HOLD Master File, use SET ASNAMES.

- All characters are converted to uppercase.
- Special characters and blanks used in the AS phrase are preserved in the field name that FOCUS creates when SET ASNAMES is used. When you refer to these non-standard field names in the newly created Master File, you must use single quotation marks around the field name.
- Text specified in an AS phrase that contains more than 66 characters is truncated to 66 characters in the Master File.
- Duplicate field names may occur in the newly created Master File as a result of truncation or the way AS phrases have been specified. In this case, refer to the fields by their aliases (E01, E02, and so forth).

- When commas are used as delimiters to break lines in the column heading, only the literal up to the first comma is used as the field name in the Master File. For example,

  ```
  PRINT COUNTRY AS 'PLACE,OF,ORIGIN'
  ```

  produces the field name, PLACE, in the HOLD Master File.

- Unless SET ASNAMES=ON has been issued, field names exceeding 12 characters are used as field names in the HOLD Master File.

- When ACROSS is used in a TABLE request and the results are put in a HOLD file, the columns generated by the ACROSS phrase all have the same field name in the HOLD Master File. If SET ASNAMES is used, each new column may have a unique field name. This unique field name consists of the ASNAME value specified in the request's display command, concatenated to the beginning of the value of the field used in the ACROSS phrase. If several field names have the same letters, this approach will not work.

  If an AS phrase is used for the fields in the ACROSS phrase, each new column will have a field name composed of the literal in the AS phrase concatenated to the beginning of the value of the first field used in the ACROSS phrase.

Example Controlling Field Names in the HOLD Master File

In the following example, SET ASNAMES=ON causes the text in the AS phrase to be used as field names in the HOLD1 Master File. The two fields in the HOLD1 Master File, NATION and AUTOMOBILE, contain the data for COUNTRY and CAR.

```
SET ASNAMES=ON
TABLE FILE CAR
PRINT CAR AS 'AUTOMOBILE'
BY COUNTRY AS 'NATION'
ON TABLE HOLD AS HOLD1
END
```

The request produces the following Master File:

```
FILE=HOLD1              ,SUFFIX=FIX
SEGNAME=HOLD1, SEGTYPE=S01,$
FIELDNAME   =NATION                ,E01       ,A10      ,A12     ,$
FIELDNAME   =AUTOMOBILE            ,E02       ,A16      ,A16     ,$
```

Example Providing Unique Field Names

The following request generates a HOLD Master File with one unique field name for SALES and one for AVE.SALES. Both SALES and AVE.SALES would be named SALES, had SET ASNAMES not been used.

```
SET ASNAMES=ON
TABLE FILE CAR
SUM SALES AND AVE.SALES AS 'AVERAGESALES'
BY CAR
ON TABLE HOLD AS HOLD2
END
```

The request produces the following Master File:

```
FILE=HOLD2       ,SUFFIX=FIX
SEGNAME=HOLD2, SEGTYPE=S01,$
FIELDNAME   =CAR             ,E01         ,A16     ,A16     ,$
FIELDNAME   =SALES           ,E02         ,I6      ,I04     ,$
FIELDNAME   =AVERAGESALES    ,E03         ,I6      ,I04     ,$
```

Example Using SET ASNAMES With the ACROSS Phrase

The following request produces a HOLD Master File with the literal CASH concatenated to each value of COUNTRY:

```
SET ASNAMES=ON
TABLE FILE CAR
SUM SALES AS 'CASH'
ACROSS COUNTRY
ON TABLE HOLD AS HOLD3
END
```

This is the Master File produced by the request:

```
FILE=HOLD3          ,SUFFIX=FIX
SEGNAME=HOLD3, SEGTYPE=S01,$
FIELDNAME   =CASHENGLAND       ,E01         ,I6       ,I04      ,$
FIELDNAME   =CASHFRANCE        ,E02         ,I6       ,I04      ,$
FIELDNAME   =CASHITALY         ,E03         ,I6       ,I04      ,$
FIELDNAME   =CASHJAPAN         ,E04         ,I6       ,I04      ,$
FIELDNAME   =CASHW GERMANY     ,E05         ,I6       ,I04      ,$
```

Without the SET ASNAMES command, every field in the HOLD FILE would be named COUNTRY.

To generate field names for ACROSS values that include only the field value, use the AS phrase followed by two single quotation marks as follows:

```
SET ASNAMES=ON
TABLE FILE CAR
SUM SALES AS ''
ACROSS COUNTRY
ON TABLE HOLD AS HOLD4
END
```

The resulting Master File looks like this:

```
FILE=HOLD4          ,SUFFIX=FIX
SEGNAME=HOLD4
FIELDNAME   =ENGLAND         ,E01         ,I6       ,I04      ,$
FIELDNAME   =FRANCE          ,E02         ,I6       ,I04      ,$
FIELDNAME   =ITALY           ,E03         ,I6       ,I04      ,$
FIELDNAME   =JAPAN           ,E04         ,I6       ,I04      ,$
FIELDNAME   =W GERMANY       ,E05         ,I6       ,I04      ,$
```

Controlling Fields in HOLD Files: SET HOLDLIST

You can use the SET command, SET HOLDLIST, to restrict fields in HOLD files to those appearing in a request.

Syntax How to Control Fields

The syntax is:

```
SET HOLDLIST = {PRINTONLY|ALL}
```

Possible values are:

`PRINTONLY`

Specifies that only those fields that would have appeared in the report are included in the generated HOLD file. Fields that are noprinted are not included in the HOLD file.

`ALL`

Specifies that all display fields referenced in a request will appear in a HOLD file, including both computed fields and fields mentioned in the COMPUTE statement. ALL is the default value. OLD may be used as a synonym for ALL.

Reference Usage Notes for Controlling Fields

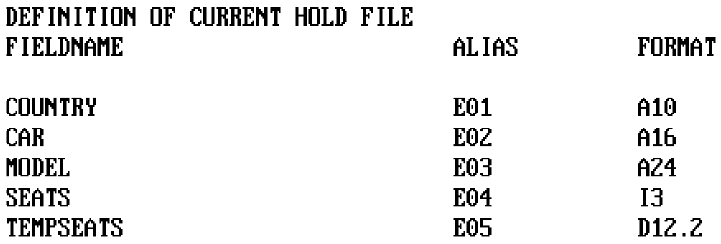
`SET HOLDLIST` may also be issued from within a TABLE request.

Note: When used with MATCH, SET HOLDLIST always behaves as if HOLDLIST is set to ALL.

Example Using HOLDLIST=ALL

When HOLDLIST is set to ALL, the following TABLE request produces a HOLD file containing all specified fields, including NOPRINT and COMPUTE fields:

```
SET HOLDLIST=ALL

TABLE FILE CAR
PRINT CAR MODEL NOPRINT
COMPUTE TEMPSEATS=SEATS+1;
BY COUNTRY
ON TABLE HOLD
END

? HOLD
```

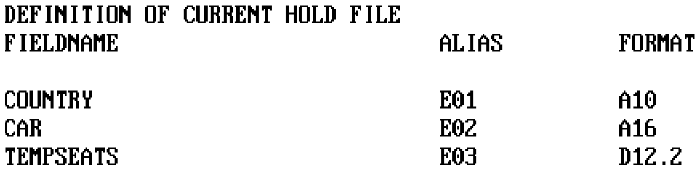


```
DEFINITION OF CURRENT HOLD FILE
FIELDNAME                         ALIAS         FORMAT

COUNTRY                           E01           A10
CAR                               E02           A16
MODEL                             E03           A24
SEATS                             E04           I3
TEMPSEATS                         E05           D12.2
```

Example Using HOLDLIST= PRINTONLY

When HOLDLIST is set to PRINTONLY, the following TABLE request produces a HOLD file containing only fields that would be displayed in TABLE output:

```
SET HOLDLIST=PRINTONLY

TABLE FILE CAR
PRINT CAR MODEL NOPRINT
COMPUTE TEMPSEATS=SEATS+1;
BY COUNTRY
ON TABLE HOLD
END

? HOLD
```



```
DEFINITION OF CURRENT HOLD FILE
FIELDNAME                         ALIAS         FORMAT

COUNTRY                           E01           A10
CAR                               E02           A16
TEMPSEATS                         E03           D12.2
```

Controlling the TITLE and ACCEPT Attributes in the HOLD Master File

The SET HOLDATTR command controls whether the TITLE and ACCEPT attributes in the original Master File are propagated in the HOLD Master File. SET HOLDATTR does not affect the way fields are named in the HOLD Master File.

Syntax How to Control TITLE and ACCEPT Attributes

The syntax is:

```
SET HOLDATTR = value
```

Possible values are:

`ON`

Uses the TITLE attribute as specified in the original Master File in HOLD files in any format. The ACCEPT attribute will be propagated to the HOLD Master File only for HOLD files in FOCUS format.

`OFF`

Does not use the TITLE or ACCEPT attributes from the original Master File in the HOLD Master File.

`FOCUS`

Uses the TITLE and ACCEPT attributes only for HOLD files in FOCUS format. This is the default.

Reference Usage Notes for Controlling TITLES and ACCEPT Attributes

If a data source field does not have the TITLE attribute specified in the Master File, but there is an AS phrase specified for the field in a report request, the corresponding field in the HOLD file will be named according to the AS phrase.

Example Controlling Master File Attributes

For example, the Master File for the CAR database specifies TITLE and ACCEPT attributes:

```
FILENAME=CAR2, SUFFIX=FOC
SEGNAME=ORIGIN, SEGTYPE=S1
  FIELDNAME =COUNTRY, COUNTRY, A10, TITLE='COUNTRY OF ORIGIN',
             ACCEPT='CANADA' OR 'ENGLAND' OR 'FRANCE' OR 'ITALY' OR
                    'JAPAN' OR 'W GERMANY',
             FIELDTYPE=I,$
SEGNAME=COMP, SEGTYPE=S1, PARENT=ORIGIN
  FIELDNAME=CAR, CARS, A16, TITLE='NAME OF CAR',$
.
.
.
```

Using SET HOLDATTR=FOCUS, the following request

```
TABLE FILE CAR2
PRINT CAR
BY COUNTRY ON TABLE HOLD FORMAT FOCUS AS HOLD5
END
```

will produce this HOLD Master File:

```
FILE=HOLD5              ,SUFFIX=FOC
SEGNAME=SEG01          ,SEGTYPE=S02
FIELDNAME =COUNTRY ,E01   ,A10
      TITLE='COUNTRY OF ORIGIN',
      ACCEPT=CANADA ENGLAND FRANCE ITALY JAPAN 'W GERMANY',$
FIELDNAME =FOCLIST    ,E02       ,I5     ,$
FIELDNAME =CAR        ,E03       ,A16    ,
      TITLE='NAME OF CAR' ,$
```

The SAVE File

A SAVE file is an extract file that saves the data of a report, but does not save headings, subtotals, or create a Master File. It is a simple sequential character data file and can be used by other programs, or merged into another data file using a FOCUS data maintenance request. The default format is simple character, although you can specify formats compatible with many other software products.

Syntax How to Create SAVE Files

There are three ways you can create a SAVE file

```
ON TABLE SAVE [AS filename] [FORMAT fmt] [MISSING {ON|OFF}]
```

```
display_field SAVE [AS filename] [FORMAT fmt] [MISSING {ON|OFF}]
```

At the FOCUS command line or in a stored procedure, issue the SAVE command with appropriate options after the report is printed.

where:

`AS` *`filename`*

Specifies a file name for your SAVE file. If you do not specify a file name, it defaults to SAVE. Each SAVE command will overwrite the previous SAVE file. Coding a distinct file name in each request will direct the saved data to its own file.

`FORMAT` *`fmt`*

Specifies the format of the SAVE extract file.

`DIF` captures the entire report output—including headings and footings—and creates a comma-delimited file for use with most spreadsheet packages.

`LOTUS` captures all the columns of the FOCUS report request in a character file which LOTUS 1-2-3 can then import. All alphanumeric fields are enclosed in quotation marks. Columns are separated by commas. Maximum record length is 512 bytes. The LOTUS file has a file name of PRN in CMS and allocates a scratch data set to the file HOLD.

`SYLK` captures all the columns of the report request in a character file for Microsoft's spreadsheet program Multiplan. The generated file cannot have more than 9,999 rows.

`WP` captures the entire report output—including headings, footings, and subtotals—and creates a text file that can be easily incorporated into most word processing packages. To include page control information in the file, specify the TABPAGENO option in a heading or the SET PAGE=OFF command. A report saved in WP format uses the character `1` in column 1 to indicate page control information. SAVE FORMAT WP is synonymous with HOLD FORMAT WP.

`MISSING`

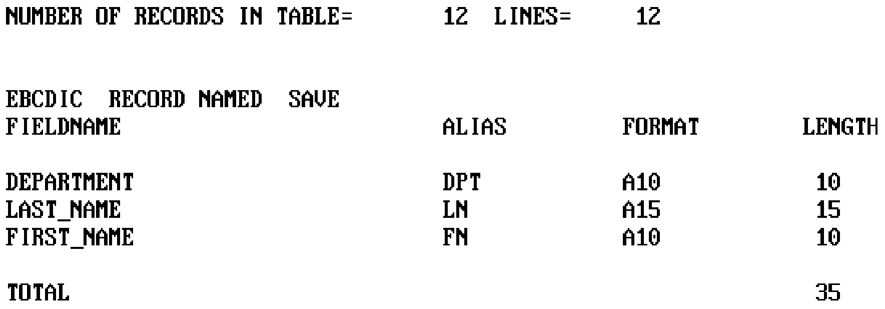
Controls whether fields with the attribute MISSING=ON in the Master File are carried over into the SAVE file. The default is MISSING OFF.

Example Creating a SAVE File

For example:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
   BY DEPARTMENT
   ON TABLE SAVE
   END
```

The request produces the following screen display, which describes the layout of the extract file:



```
NUMBER OF RECORDS IN TABLE=       12  LINES=     12


EBCDIC  RECORD NAMED  SAVE
FIELDNAME                         ALIAS         FORMAT        LENGTH

DEPARTMENT                        DPT           A10            10
LAST_NAME                         LN            A15            15
FIRST_NAME                        FN            A10            10

TOTAL                                                          35
```

Note: Due to limitations to the use of text fields with FIXFORM (See the *Maintaining Databases* manual), the following restrictions apply when using text fields with the SAVE option:

- When you create a SAVE file, you can include as many text fields in the file as needed. However, you must specify text fields after non-text fields in the display list (PRINT…, SUM…, and so forth).
- When you create a SAVE file with a request that includes an ACROSS phrase, you can specify only one text field in the display list, and no non-text fields.

The SAVB File

The SAVB file is like a SAVE file in that no Master File is created, but the data file is in internal format (described earlier as the default format of a HOLD file).

Syntax How to Create SAVB Files

There are three ways to create a SAVB file,

ON TABLE SAVB [AS *filename*] [MISSING {ON|OFF}]

display_field SAVB [AS *filename*] [MISSING {ON|OFF}]

or, at the FOCUS command line, issue SAVB with appropriate options after the report is printed.

where:

AS *filename*

Renames the file. The default is SAVB. Unless you give it a name, subsequent SAVB files will overlay each other. Coding a distinct file name in each request will direct the data to its own file.

MISSING

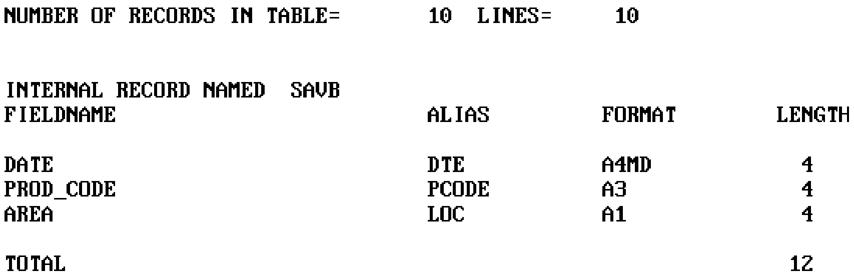
Ensures that fields with the attribute, MISSING ON, will be carried over into the HOLD file. The default is MISSING OFF.

Example Creating a SAVB File

For example:

```
TABLE FILE SALES
PRINT PROD_CODE AND AREA
BY DATE
WHERE CITY IS 'STAMFORD' OR 'UNIONDALE'
ON TABLE SAVB
END
```

A description of the binary extract file is displayed after the records are retrieved. It appears as:



```
NUMBER OF RECORDS IN TABLE=       10  LINES=     10


INTERNAL RECORD NAMED  SAVB
FIELDNAME                         ALIAS         FORMAT        LENGTH

DATE                              DTE           A4MD            4
PROD_CODE                         PCODE         A3              4
AREA                              LOC           A1              4

TOTAL                                                          12
```

Note: Due to limitations on the use of text fields with FIXFORM (see the *Maintaining Databases* manual), the following restrictions apply when using text fields with the SAVB option.

- When you create a SAVB file, you can include as many text fields in the file as needed. However, you must specify text fields after non-text fields in the display list (PRINT…, SUM…,and so forth).
- When you create a SAVB file with a request that includes an ACROSS phase, you can specify only one text field in the display list, and no non-text fields.

13 Handling Records With Missing Field Values

Topics:

- Irrelevant Report Data
- Missing Field Values
- Handling Missing Segment Instances

Missing data is defined as data that is missing from a report because it is not relevant or because it does not exist in the database. Report results that involve averaging and counting calculations or the display of parent segment instances may be affected. Data can be missing from reports and calculations for three reasons:

- Data is not relevant to a particular row and column in a report. For an example of irrelevant report data, see *Irrelevant Report Data* on page 13-3.
- A field in a segment instance does not have a data value. For more information, see *Missing Field Values* on page 13-4.

  If a segment instance contains values for some fields but not for others, FOCUS marks the omitted values with a special internal code to distinguish the missing values from blanks or zeros. The special internal code is supplied when you specify the MISSING ON attribute for database and temporary fields.

  Missing data for a field in a segment instance may occur when the data values are unknown. For example, consider the EMPLOYEE database. Suppose the employees are due for a pay raise by a certain date, but the amount of the raise has not been determined. The company enters the date for each employee into the database without the salary amounts; the salaries will be entered later. Each date is an individual instance in the salary history segment, but the new salary for each date instance is missing.

  In addition, suppose a request averages the SALARY field (SUM AVE.SALARY). The resulting average depends on whether the missing values for the SALARY field are treated as zeros or as internal codes. Missing data also affects the calculations of DEFINE commands.

- A parent segment instance does not have child instances (missing segment instances). For more information, see *Handling Missing Segment Instances* on page 13-14.

  In multi-segment files, when an instance in a parent segment does not have descendant instances, the nonexistent descendant instances are called missing instances. The retrieval path, in this case, is called a short path. FOCUS never considers unique segments to be missing.

  For example, consider the subset of the EMPLOYEE file (see Appendix A, *Master Files and Diagrams)*. The top segment contains employee ID numbers and names. The bottom segment contains each employee’s salary history: the date the employee was granted a new salary and the amount of the salary.

  Suppose some employees are paid by an outside agency. None of these employees have a company salary history. Instances referring to these employees in the salary history segment are missing.

  Nonexistent descendant instances affect whether parent segment instances are included in report results. The SET ALL command and the ALL. prefix enable you to include parent segment data in reports.

  Note: To run the examples in this section, use stored procedures EMPMISS and SALEMISS. These stored procedures add missing data to the existing EMPLOYEE and SALES databases, respectively.

Irrelevant Report Data

Data can be missing from a report because it is not relevant. The missing or inapplicable value is indicated by the NODATA default character, a period. You may specify a more meaningful NODATA value by issuing the SET NODATA command (see *Setting the NODATA Character String* on page 13-5).

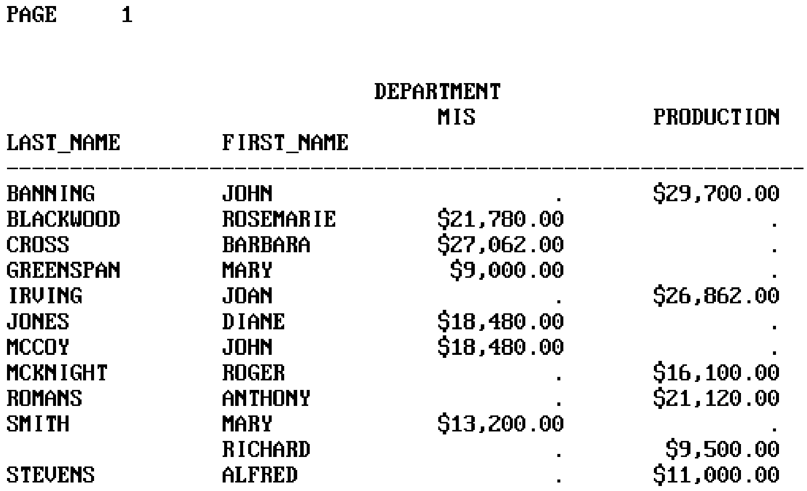
Example Irrelevant Report Data

Suppose this request displays employee salaries. The rows display employee’s last and first names; the columns represent the departments.

```
TABLE FILE EMPLOYEE
PRINT SALARY
BY LAST_NAME
BY FIRST_NAME
ACROSS DEPARTMENT
END
```

The salary for an employee working in the production department appears in the PRODUCTION column, not in the MIS column. The corresponding value in the MIS column is missing because the salary appears only under the department where the person is employed.

The chart appears as shown:



```
PAGE     1


                             DEPARTMENT
                                  MIS              PRODUCTION
LAST_NAME        FIRST_NAME
-------------------------------------------------------------
BANNING          JOHN                      .       $29,700.00
BLACKWOOD        ROSEMARIE        $21,780.00                .
CROSS            BARBARA          $27,062.00                .
GREENSPAN        MARY              $9,000.00                .
IRVING           JOAN                      .       $26,862.00
JONES            DIANE            $18,480.00                .
MCCOY            JOHN             $18,480.00                .
MCKNIGHT         ROGER                     .       $16,100.00
ROMANS           ANTHONY                   .       $21,120.00
SMITH            MARY             $13,200.00                .
                 RICHARD                   .        $9,500.00
STEVENS          ALFRED                    .       $11,000.00
```

Missing Field Values

Missing values within segment instances occur when the instances exist, but some fields in the instances lack values.

When fields in instances lack values, FOCUS assigns numeric fields the value of 0, and alphanumeric fields the value of blank. These default values appear in reports and are used in all calculations performed by the SUM and COUNT display commands, DEFINE commands, and prefix operators such as MAX. and AVE.

To prevent the use of these default values in calculations (which might then give erroneous results), FOCUS provides the MISSING ON attribute. This attribute causes FOCUS to supply a special internal code and to ignore the missing values. In reports, the special internal code is represented with the SET NODATA value.

Example Counting With Missing Values

For example, suppose the field CURR_SAL appears in 12 segment instances. In three of those instances, the field was given no value. The TABLE command

```
COUNT CURR_SAL
```

counts 12 occurrences of the CURR_SAL field anyway.

Example Averaging With Missing Values

For example, suppose you had the following records of data for a certain field:

```
.
.
1
3
```

The numeric values in the first two records are missing (indicated by the periods); the last two records have values of 1 and 3. If you average these fields without supplying the MISSING attribute (MISSING OFF), FOCUS automatically supplies a value of 0 for the two records missing values. Thus, the average of these four records is (0+0+1+3)/ 4, or 1. If you use the MISSING ON attribute, FOCUS ignores the two records missing values, calculating the average as (1+3)/2, or 2.

Setting the NODATA Character String

The NODATA character string marks places in the report where no data or inapplicable data appears. To set the NODATA string, enter the following command:

```
SET NODATA = string
```

The default NODATA string is a period (.). The string may be up to 11 characters long. Common choices are NONE, N/A, and MISSING.

If you do not want any characters, issue:

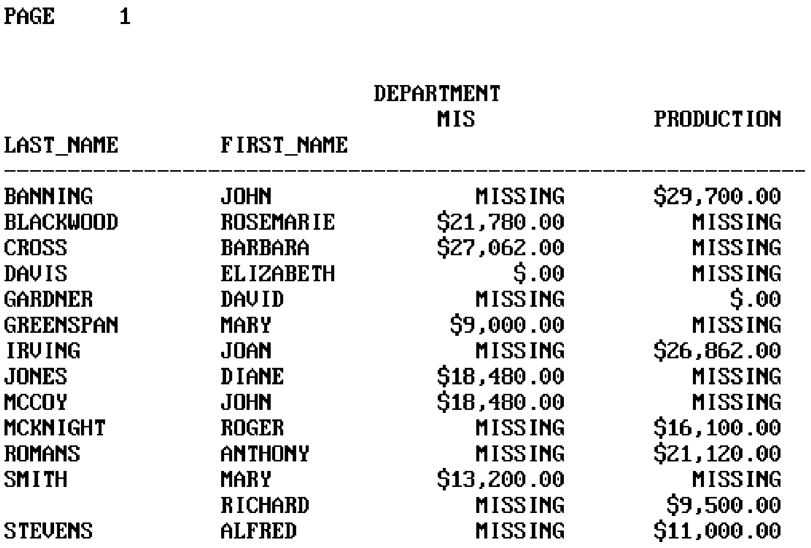
```
SET NODATA = ' '
```

Example Setting the NODATA Character String

For instance, consider the following request:

```
SET NODATA=MISSING

TABLE FILE EMPLOYEE
PRINT CURR_SAL BY LAST_NAME BY FIRST_NAME
ACROSS DEPARTMENT
END
```

The request produces the following report. Since the NODATA character string is set to MISSING, the word MISSING appears on the report instead of the period default.



```
PAGE     1


                             DEPARTMENT
                                  MIS              PRODUCTION
LAST_NAME        FIRST_NAME
--------------------------------------------------------------
BANNING          JOHN                MISSING       $29,700.00
BLACKWOOD        ROSEMARIE        $21,780.00          MISSING
CROSS            BARBARA          $27,062.00          MISSING
DAVIS            ELIZABETH              $.00          MISSING
GARDNER          DAVID               MISSING             $.00
GREENSPAN        MARY              $9,000.00          MISSING
IRVING           JOAN                MISSING       $26,862.00
JONES            DIANE            $18,480.00          MISSING
MCCOY            JOHN             $18,480.00          MISSING
MCKNIGHT         ROGER               MISSING       $16,100.00
ROMANS           ANTHONY             MISSING       $21,120.00
SMITH            MARY             $13,200.00          MISSING
                 RICHARD             MISSING        $9,500.00
STEVENS          ALFRED              MISSING       $11,000.00
```

The MISSING Attribute

In some applications, you may want FOCUS to use the default values (blanks and zeros), for it may represent valid data rather than the absence of information. However, if this is not the case, use the MISSING attribute.

The MISSING attribute is placed in the Master File in the declaration of the field missing the values, after the format. For example, this field declaration specifies the MISSING attribute for the RETURNS field:

```
FIELDNAME=RETURNS, ALIAS=RTN, FORMAT=I4, MISSING=ON,$
```

When a field has the MISSING ON attribute, FOCUS marks the absence of data for the field with a special internal code rather than with blanks or zeros.

During report generation, the SUM and COUNT display commands, and all prefix commands (AVE., MAX., or MIN.) will not include the missing data in their computation when the field has the MISSING attribute in the Master File.

Note: You may add MISSING field attributes to the Master File at any time. However, MISSING attributes affect only data entered into the database after the attributes were added. For complete information about the MISSING attribute and field declarations, see the *Describing Data* manual.

Example Using the Missing Attribute

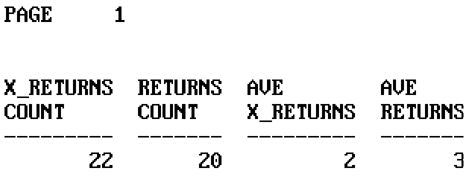
Suppose you set a DEFINE field equal to a field with the MISSING attribute. When the field with the MISSING attribute is missing a value, the corresponding DEFINE field value is 0 or blank (see *Missing Values in DEFINE Commands* on page 13-7). The DEFINE command is as follows:

```
DEFINE FILE SALES
X_RETURNS/I4 = RETURNS;
END
```

The field X_RETURNS has the same value as RETURNS except when RETURNS is missing a value, in which case the X_RETURNS value is 0. Now enter the following request:

```
TABLE FILE SALES
SUM CNT.X_RETURNS CNT.RETURNS AVE.X_RETURNS AVE.RETURNS
END
```

The request produces the following report:



```
PAGE     1


X_RETURNS  RETURNS  AVE        AVE
COUNT      COUNT    X_RETURNS  RETURNS
---------  -------  ---------  -------
       22       20          2        3
```

The count for the RETURNS field is lower than the count for X_RETURNS and the average for RETURNS is higher than for X_RETURNS, because the missing values in RETURNS are not part of the calculations.

Missing Values in DEFINE Commands

A DEFINE expression can contain fields that have missing values. When a field is missing a value, FOCUS evaluates the field as 0 or blank for the purpose of computation, even if the field has the MISSING attribute.

You can use the MISSING attribute in:

- DEFINE expressions in DEFINE commands.
- DEFINE attributes in Master Files.

Note: You cannot use the MISSING attribute in COMPUTE phrases in TABLE report requests.

Syntax The MISSING Attribute in DEFINE Commands

To issue a DEFINE command whose expression includes fields that have missing values, use the following syntax

```
field[/format] MISSING {ON|OFF} [NEEDS] {SOME|ALL} [DATA] = expression;
```

where:

field
Is the DEFINE field.

/format
Is the format of the field. The default is D12.2.

MISSING ON
Enables FOCUS to declare the value of the defined or computed field to be missing. The default is OFF.

NEEDS
Is optional. It helps to clarify the meaning of the command.

SOME
Is the default. Indicates that if at least one field in the expression has a value, the DEFINE field has a value (the field’s missing values are evaluated as 0 or blank). If all of the fields in the expression are missing values, the DEFINE field is missing its value.

ALL
Indicates that if all the fields in the expression have values, the defined or computed field has a value. If at least one field in the expression is missing a value, the defined or computed field is missing its value.

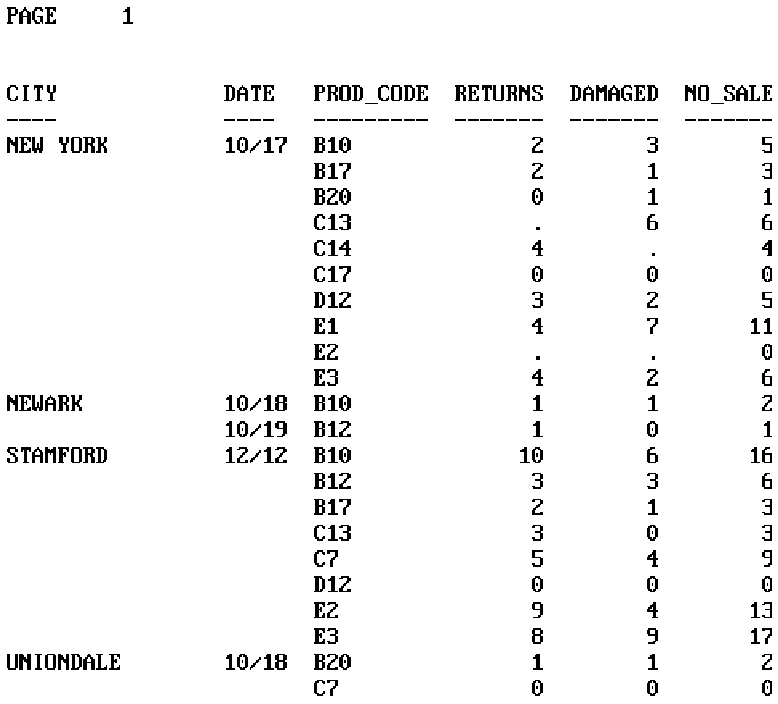
DATA
Is optional. It helps to clarify the meaning of the command.

Example Missing Values in DEFINE FIELDS

The following request uses the two fields, RETURNS and DAMAGED, to define NO_SALE. Both the RETURNS and DAMAGED fields have the MISSING attribute in the SALES Master File, yet whenever one of these fields is missing a value, FOCUS evaluates that field as 0. The request is:

```
DEFINE FILE SALES
NO_SALE/I4 = RETURNS + DAMAGED;
END

TABLE FILE SALES
PRINT RETURNS AND DAMAGED AND NO_SALE
BY CITY BY DATE BY PROD_CODE
END
```

The request produces the following report. Notice the products C13, C14, and E2 in the New York section.



```
PAGE     1


CITY              DATE   PROD_CODE  RETURNS  DAMAGED  NO_SALE
----              ----   ---------  -------  -------  -------
NEW YORK          10/17  B10              2        3        5
                         B17              2        1        3
                         B20              0        1        1
                         C13              .        6        6
                         C14              4        .        4
                         C17              0        0        0
                         D12              3        2        5
                         E1               4        7       11
                         E2               .        .        0
                         E3               4        2        6
NEWARK            10/18  B10              1        1        2
                  10/19  B12              1        0        1
STAMFORD          12/12  B10             10        6       16
                         B12              3        3        6
                         B17              2        1        3
                         C13              3        0        3
                         C7               5        4        9
                         D12              0        0        0
                         E2               9        4       13
                         E3               8        9       17
UNIONDALE         10/18  B20              1        1        2
                         C7               0        0        0
```

Example Using SOME and ALL in DEFINEd Fields

You can specify that if either some or all of the field values in the DEFINE expression are missing, the new defined field should also be missing its value by using the syntax shown previously. For example, the following DEFINE command creates the fields SOMEDATA and ALLDATA.

```
DEFINE FILE SALES
SOMEDATA/I5 MISSING ON NEEDS SOME=RETURNS + DAMAGED;
ALLDATA/I5 MISSING ON NEEDS ALL=RETURNS + DAMAGED;
END

TABLE FILE SALES
PRINT RETURNS AND DAMAGED SOMEDATA ALLDATA
BY CITY BY DATE BY PROD_CODE
END
```

where:

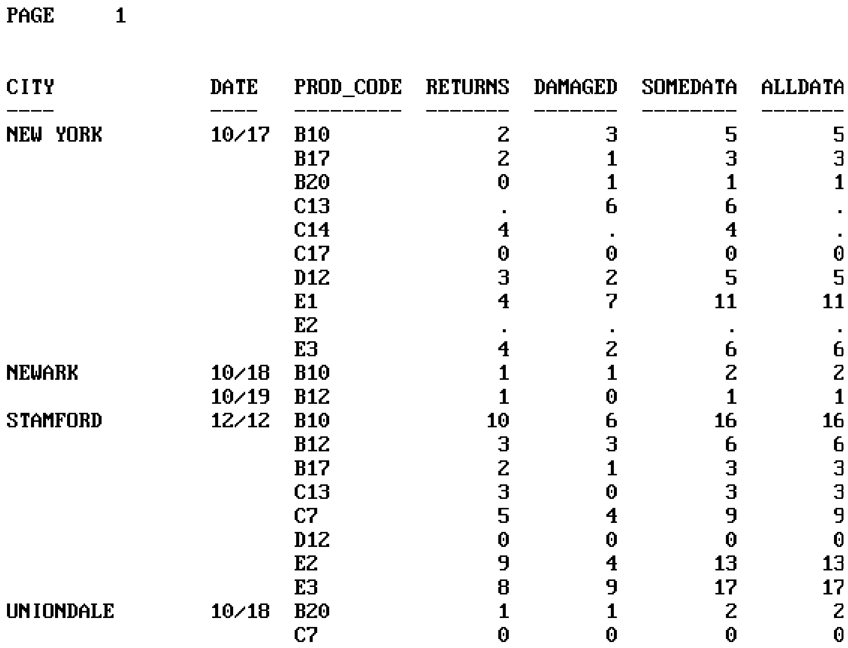
`SOMEDATA`

Contains a value if either the RETURNS or DAMAGED field contains a value. Otherwise, SOMEDATA is missing its value.

`ALLDATA`

Contains a value only if both the RETURNS and DAMAGED fields contain values. Otherwise, ALLDATA is missing its value.

The request produces the following report:



```
PAGE     1


CITY            DATE   PROD_CODE  RETURNS  DAMAGED  SOMEDATA  ALLDATA
----            ----   ---------  -------  -------  --------  -------
NEW YORK        10/17  B10              2        3         5        5
                       B17              2        1         3        3
                       B20              0        1         1        1
                       C13              .        6         6        .
                       C14              4        .         4        .
                       C17              0        0         0        0
                       D12              3        2         5        5
                       E1               4        7        11       11
                       E2               .        .         .        .
                       E3               4        2         6        6
NEWARK          10/18  B10              1        1         2        2
                10/19  B12              1        0         1        1
STAMFORD        12/12  B10             10        6        16       16
                       B12              3        3         6        6
                       B17              2        1         3        3
                       C13              3        0         3        3
                       C7               5        4         9        9
                       D12              0        0         0        0
                       E2               9        4        13       13
                       E3               8        9        17       17
UNIONDALE       10/18  B20              1        1         2        2
                       C7               0        0         0        0
```

Testing for Segments With Missing Field Values

You can use WHERE to identify segment instances missing field values. You cannot use these tests to identify missing instances. To test for missing instances using the ALL PASS parameter, see *Handling Missing Segment Instances* on page 13-14.

The WHERE syntax is:

```
WHERE field {IS|EQ} MISSING
```

They are equivalent, and will produce the same result.

To test whether field values are present, use:

```
WHERE field {NE|IS-NOT} MISSING
```

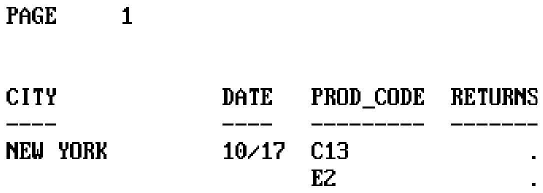
A WHERE clause that tests a numeric field for 0 or an alphanumeric field for blanks also tests the field for missing values.

Example Testing for Missing Field Values

For example, this first request displays grocery items (by code) for which the number of packages returned by customers is not recorded (or missing).

```
TABLE FILE SALES
PRINT RETURNS
BY CITY BY DATE BY PROD_CODE
WHERE RETURNS IS MISSING
END
```

The request generates this report:



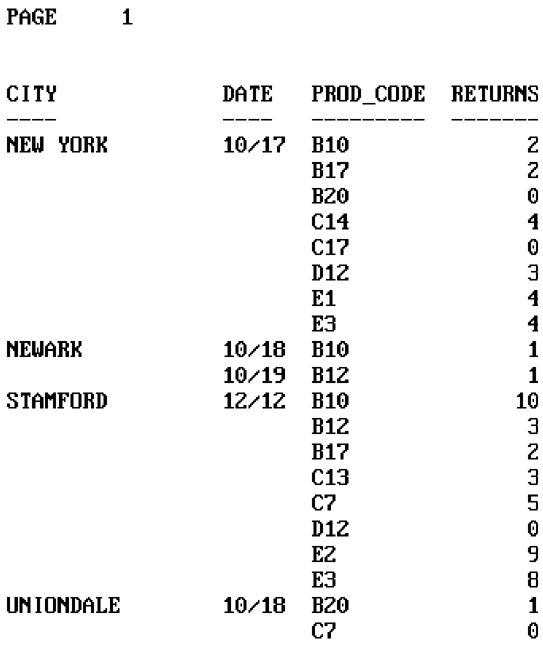
```
PAGE     1


CITY            DATE   PROD_CODE  RETURNS
----            ----   ---------  -------
NEW YORK        10/17  C13              .
                       E2               .
```

Example Testing for Present Field Values

This request displays only those grocery items for which the number of packages returned by customers is not missing.

```
TABLE FILE SALES
PRINT RETURNS
BY CITY BY DATE BY PROD_CODE
WHERE RETURNS IS-NOT MISSING
END
```

The request generates the report below:



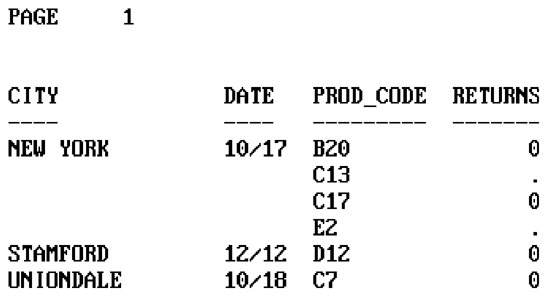
```
PAGE     1


CITY             DATE   PROD_CODE  RETURNS
----             ----   ---------  -------
NEW YORK         10/17  B10              2
                        B17              2
                        B20              0
                        C14              4
                        C17              0
                        D12              3
                        E1               4
                        E3               4
NEWARK           10/18  B10              1
                 10/19  B12              1
STAMFORD         12/12  B10             10
                        B12              3
                        B17              2
                        C13              3
                        C7               5
                        D12              0
                        E2               9
                        E3               8
UNIONDALE        10/18  B20              1
                        C7               0
```

Example Testing for Blanks or Zeros

This request displays grocery items that either were never returned or for which the number of packages was never recorded:

```
TABLE FILE SALES
PRINT RETURNS
BY CITY BY DATE BY PROD_CODE
WHERE RETURNS EQ 0
END
```

It generates the following report:

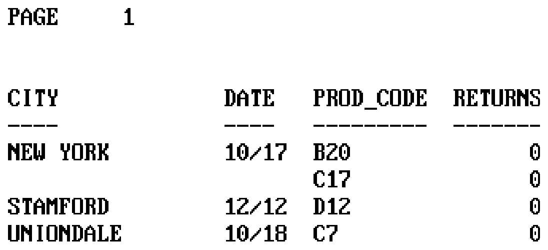


```
PAGE     1


CITY             DATE   PROD_CODE  RETURNS
----             ----   ---------  -------
NEW YORK         10/17  B20              0
                        C13              .
                        C17              0
                        E2               .
STAMFORD         12/12  D12              0
UNIONDALE        10/18  C7               0
```

To print only those items that have not been returned by customers, you need two WHERE clauses: one to restrict the number of returns to 0, the other to exclude missing values, as in the following request:

```
TABLE FILE SALES
PRINT RETURNS
BY CITY BY DATE BY PROD_CODE
WHERE RETURNS EQ 0
WHERE RETURNS IS-NOT MISSING
END
```

The request produces the following report:



```
PAGE     1


CITY             DATE   PROD_CODE  RETURNS
----             ----   ---------  -------
NEW YORK         10/17  B20              0
                        C17              0
STAMFORD         12/12  D12              0
UNIONDALE        10/18  C7               0
```

Missing Data Values Preserved in Extract Files

The ability to distinguish between missing data and default values (blanks and zeros) in fields can be carried over into extract files (extract files are explained in Chapter 12, *Saving and Reusing Your Report Output).* If the retrieved and processed information would have displayed the NODATA string in a report, FOCUS can store the NODATA string in the extract file. To prepare such an extract file, use this syntax in the TABLE request,

```
ON TABLE extract_type MISSING {ON|OFF}
```

where:

extract_type

Specifies the type of extract file.

HOLD creates an extract file, which can be used for subsequent reports. The default value for MISSING is ON.

SAVE creates a text extract file, which can be used by other programs. The default for MISSING is OFF.

SAVB creates a binary extract file, which can be used by other programs. The default for MISSING is OFF.

HOLD files can be created with both the MISSING and FORMAT ALPHA options, specified in any order. For example:

```
ON TABLE HOLD FORMAT ALPHA MISSING OFF
```

```
ON TABLE HOLD MISSING OFF FORMAT ALPHA
```

The HOLD, SAVE, and SAVB commands can incorporate the MISSING option with display commands. For example:

```
TABLE FILE SALES
SUM RETURNS AND SAVE MISSING ON
BY CITY BY DATE BY PROD_CODE
END
```

Handling Missing Segment Instances

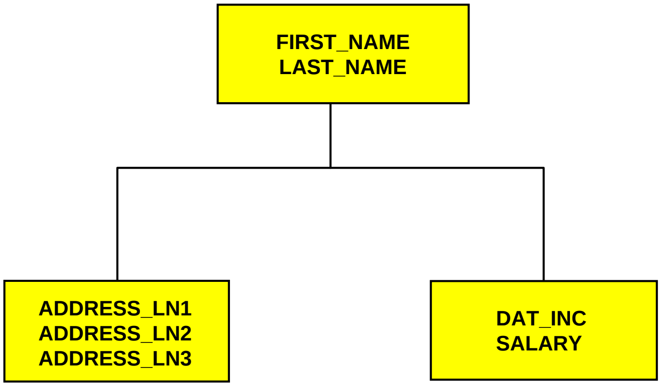
When you write a report request from a database that has missing segment instances, the missing instances affect the report. For example, if the request names fields in a segment and its descendants, the report omits parent segment instances which have no descendants. It makes no difference whether fields are display fields or sort fields.

When an instance is missing descendants in a child segment, the instance, its parent, the parent of its parent, and so on up to the root segment, is called a short path.

Example Understanding Missing Segment Instances

For example, consider a multi-segment subset of the EMPLOYEE file:

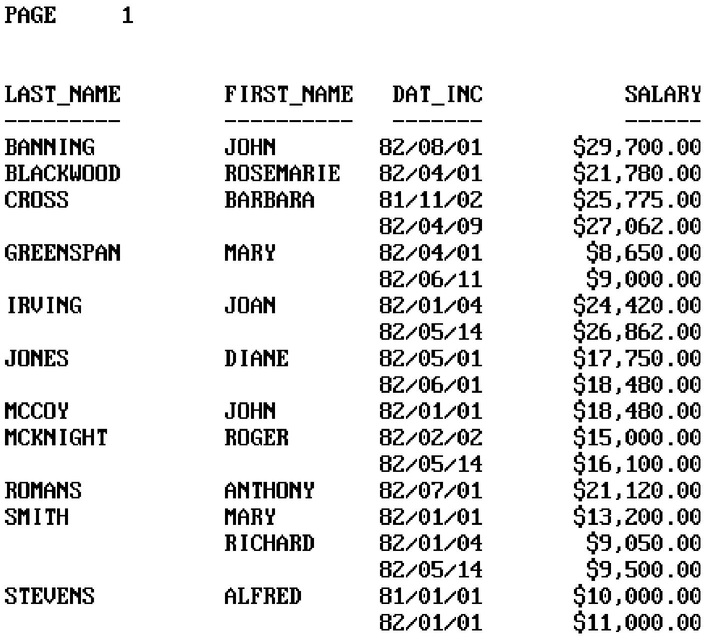
- The top segment contains employee names.
- The left segment contains addresses.
- The right segment contains salary histories.



This request prints the salary histories for each employee:

```
TABLE FILE EMPLOYEE
PRINT SALARY
BY LAST_NAME BY FIRST_NAME
BY DAT_INC
END
```

Employees Elizabeth Davis and David Gardner have no salary histories, and so are not included in the report, as shown below:



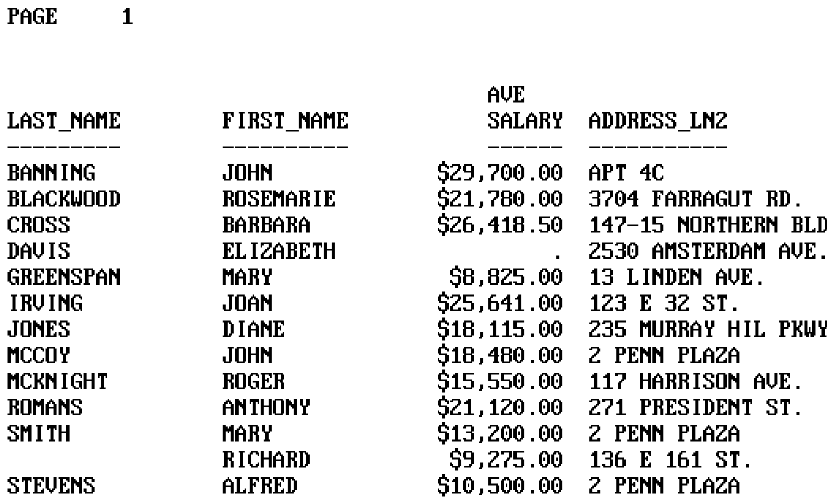
```
PAGE     1


LAST_NAME        FIRST_NAME   DAT_INC          SALARY
---------        ----------   -------          ------
BANNING          JOHN        82/08/01      $29,700.00
BLACKWOOD        ROSEMARIE   82/04/01      $21,780.00
CROSS            BARBARA     81/11/02      $25,775.00
                             82/04/09      $27,062.00
GREENSPAN        MARY        82/04/01       $8,650.00
                             82/06/11       $9,000.00
IRVING           JOAN        82/01/04      $24,420.00
                             82/05/14      $26,862.00
JONES            DIANE       82/05/01      $17,750.00
                             82/06/01      $18,480.00
MCCOY            JOHN        82/01/01      $18,480.00
MCKNIGHT         ROGER       82/02/02      $15,000.00
                             82/05/14      $16,100.00
ROMANS           ANTHONY     82/07/01      $21,120.00
SMITH            MARY        82/01/01      $13,200.00
                 RICHARD     82/01/04       $9,050.00
                             82/05/14       $9,500.00
STEVENS          ALFRED      81/01/01      $10,000.00
                             82/01/01      $11,000.00
```

Davis and Gardner are omitted because the LAST_NAME and FIRST_NAME fields belong to the root segment, and the DAT_INC and SALARY fields belong to the descendant salary history segment. Since Davis and Gardner have no descendant instances in the salary history segment, they are omitted from the report.

This next request prints the average salary and second address line of each employee. The database contains Davis' address, but not Gardner's. The request is:

```
TABLE FILE EMPLOYEE
SUM AVE.SALARY
AND ADDRESS_LN2
BY LAST_NAME BY FIRST_NAME
END
```

The request produces the following report:



```
PAGE     1


                                     AVE
LAST_NAME        FIRST_NAME          SALARY  ADDRESS_LN2
---------        ----------          ------  -----------
BANNING          JOHN            $29,700.00  APT 4C
BLACKWOOD        ROSEMARIE       $21,780.00  3704 FARRAGUT RD.
CROSS            BARBARA         $26,418.50  147-15 NORTHERN BLD
DAVIS            ELIZABETH                .  2530 AMSTERDAM AVE.
GREENSPAN        MARY             $8,825.00  13 LINDEN AVE.
IRVING           JOAN            $25,641.00  123 E 32 ST.
JONES            DIANE           $18,115.00  235 MURRAY HIL PKWY
MCCOY            JOHN            $18,480.00  2 PENN PLAZA
MCKNIGHT         ROGER           $15,550.00  117 HARRISON AVE.
ROMANS           ANTHONY         $21,120.00  271 PRESIDENT ST.
SMITH            MARY            $13,200.00  2 PENN PLAZA
                 RICHARD          $9,275.00  136 E 161 ST.
STEVENS          ALFRED          $10,500.00  2 PENN PLAZA
```

This report prints Davis' name even though Davis has no salary history, because Davis has an instance in the descendant address segment. The report omits Gardner entirely, because Gardner has neither a salary history nor an address.

Including Missing Instances in Reports: The ALL. Prefix

If a request excludes parent segment instances that lack descendants, you can include the parent instances by attaching the ALL. prefix to one of the fields in the parent segment.

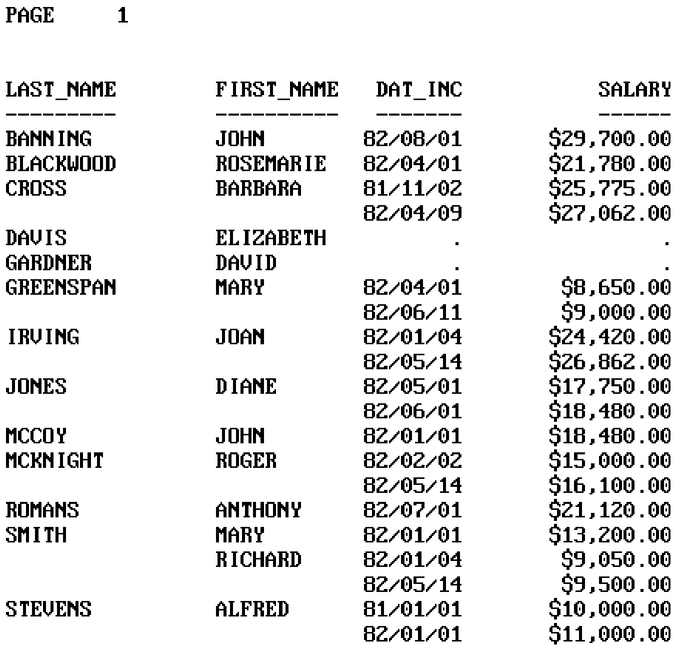
Note: If the request contains WHERE or IF commands that screen fields in segments that are missing instances, the report omits parent instances even when you use the ALL. prefix. To include the instances, use the SET ALL=PASS command described in *Including Missing Instances in Reports: The SET ALL Command* on page 13-18.

Example Including Missing Segment Instances With the ALL. Prefix

For example:

```
TABLE FILE EMPLOYEE
PRINT SALARY
BY ALL.LAST_NAME BY FIRST_NAME
BY DAT_INC
END
```

The request produces the following report:



```
PAGE     1


LAST_NAME        FIRST_NAME   DAT_INC          SALARY
---------        ----------   -------          ------
BANNING          JOHN        82/08/01      $29,700.00
BLACKWOOD        ROSEMARIE   82/04/01      $21,780.00
CROSS            BARBARA     81/11/02      $25,775.00
                             82/04/09      $27,062.00
DAVIS            ELIZABETH          .               .
GARDNER          DAVID              .               .
GREENSPAN        MARY        82/04/01       $8,650.00
                             82/06/11       $9,000.00
IRVING           JOAN        82/01/04      $24,420.00
                             82/05/14      $26,862.00
JONES            DIANE       82/05/01      $17,750.00
                             82/06/01      $18,480.00
MCCOY            JOHN        82/01/01      $18,480.00
MCKNIGHT         ROGER       82/02/02      $15,000.00
                             82/05/14      $16,100.00
ROMANS           ANTHONY     82/07/01      $21,120.00
SMITH            MARY        82/01/01      $13,200.00
                 RICHARD     82/01/04       $9,050.00
                             82/05/14       $9,500.00
STEVENS          ALFRED      81/01/01      $10,000.00
                             82/01/01      $11,000.00
```

Including Missing Instances in Reports: The SET ALL Command

You can also include parent instances with missing descendants by entering the SET ALL command before executing the request. The syntax is

```
SET ALL= include_type
```

where:

include_type

Indicates how to parent report instances with missing child instances.

`OFF` omits parent instances that are missing descendants from the report. It is the default.

`ON` includes parent instances that are missing descendants in the report. It is comparable to the ALL. prefix.

`PASS` includes parent instances that are missing descendants, even if WHERE or IF commands exist to screen fields in the descendant segments that are missing instances.

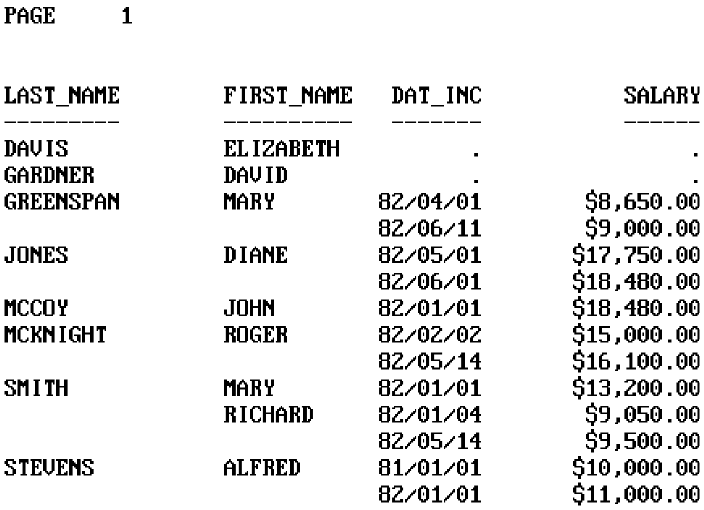
A request with WHERE or IF commands that screen fields in a segment that is missing instances omits instances in the parent segment even if you enter the SET ALL=ON command. To include these instances in the report, enter the ALL=PASS command.

Example Including Missing Segment Instances With SET ALL

For example, consider this request which includes employees without salary histories, because the ALL=PASS command is set. If the ALL=ON command is used, the employees without salary histories are omitted because of the WHERE command.

```
SET ALL=PASS

TABLE FILE EMPLOYEE
PRINT SALARY
BY LAST_NAME BY FIRST_NAME
BY DAT_INC
WHERE SALARY LT 20000
END
```

This request produces the following report:



```
PAGE     1


LAST_NAME        FIRST_NAME   DAT_INC          SALARY
---------        ----------   -------          ------
DAVIS            ELIZABETH          .               .
GARDNER          DAVID              .               .
GREENSPAN        MARY         82/04/01      $8,650.00
                              82/06/11      $9,000.00
JONES            DIANE        82/05/01     $17,750.00
                              82/06/01     $18,480.00
MCCOY            JOHN         82/01/01     $18,480.00
MCKNIGHT         ROGER        82/02/02     $15,000.00
                              82/05/14     $16,100.00
SMITH            MARY         82/01/01     $13,200.00
                 RICHARD      82/01/04      $9,050.00
                              82/05/14      $9,500.00
STEVENS          ALFRED       81/01/01     $10,000.00
                              82/01/01     $11,000.00
```

Testing for Missing Instances in FOCUS Files

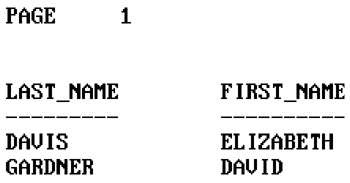
You can also use the ALL PASS parameter to produce reports that include only parent instances with missing descendant values. To do so, write the request to screen out all existing instances in the segment with missing instances. After you set the ALL parameter to PASS, the report will display only the parent instances missing descendants.

Example Testing for MISSING Instances

For example, this request screens all salary instances, since no SALARY can be both equal to 0 and not equal to 0.

```
SET ALL = PASS

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME
WHERE SALARY EQ 0
WHERE SALARY NE 0
END
```

The request produces the following report:



```
PAGE     1


LAST_NAME        FIRST_NAME
---------        ----------
DAVIS            ELIZABETH
GARDNER          DAVID
```

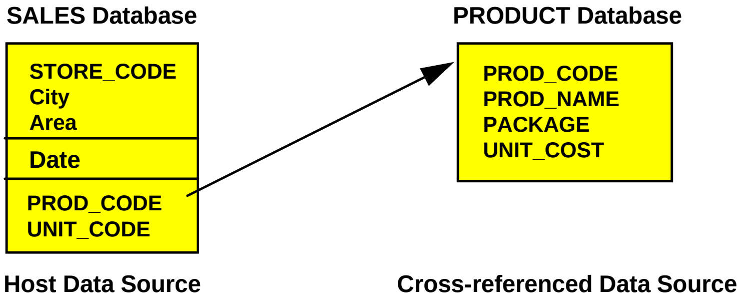
14 Joining Data Files

Topics:

- Using the JOIN Command
- Joining Fields With Different Datatypes
- Unique and Non-Unique JOIN Structures
- Data Formats of Shared Fields in Host and Cross-Referenced Files
- Restrictions on Joining Files
- Recursive JOIN Structures
- JOIN Structure Considerations
- Listing JOIN Structures: The ? JOIN Query
- Clearing JOIN Structures: The JOIN CLEAR Command

FOCUS provides a variety of features for customizing tabular reports. This chapter describes how to join data sources to gather data for your reports. You can join two or more related data sources to create a larger integrated data structure from which you can report. The joined structure is virtual—it is a way of accessing two data files as if they were a single data file—and lasts for the duration of the FOCUS session.

For example, consider the SALES and PRODUCT databases. Each store record in SALES may contain many instances of the PROD_CODE field. It would be redundant to store the associated product information with each instance of the product code; instead, PROD_CODE in SALES is joined to PROD_CODE in the PRODUCT database. PRODUCT contains a single instance of each product code and product information, saving space and making maintenance of product information easier. The joined structure is illustrated below:



You can increase retrieval speed by using an external index. However, the target segment for the index cannot be a cross-referenced segment (see Chapter 16, *Improving Data Retrieval)*.

Using the JOIN Command

The JOIN command enables you to report from two or more related data sources with a single request. Joined data sources remain physically separate, but FOCUS treats them as one database for the remainder of the FOCUS session.

With the exception of comma-delimited files, you can join any of the data source types accessible to FOCUS, which includes FOCUS, VSAM, ISAM, IMS, SQL/DS, DB2, Oracle, Teradata, Sybase, Informix, CA-DATACOM/DB, Model 204, ADABAS, CA-IDMS/DB, TOTAL, SUPRA, and simple fixed sequential files, among others.

Note: Data sources protected by the DBA security feature may not be joined from unprotected (non-DBA) data sources. When joining data sources protected by DBA security, the data sources may possess different DBA passwords.

After you issue the JOIN command, you can enter a single TABLE, TABLEF, or GRAPH request that specifies the first data source to produce a report from two or more data sources. The JOIN command also allows you to read (but not to modify) data in a second FOCUS database in a MODIFY request using the LOOKUP function. To modify multiple FOCUS databases in one request, use the COMBINE command. The LOOKUP function and the COMBINE command are described in the *Maintaining Databases* manual.

Up to 16 JOINs can be in effect at any one time. Joined data sources remain joined until the end of the FOCUS session, or until you clear the JOIN connecting the data sources.

Two data sources can be joined when they have two fields—one in each data source—that have formats in common. The JOIN is effective when the fields have values in common. For example, you need to read data from two data sources: a data source named JOB containing job information, and a data source named SALARY containing salary information. You can join these two data sources if both data sources have a field identifying the same group of employees. The data sources must identify the employees in the same way: by last name, serial number, or social security number.

When two data sources are joined, one data source is called the host file; the other data source is called the cross-referenced file. Each time FOCUS retrieves a record from the host file, it identifies the data value that should appear in the cross-referenced file. It then retrieves the records in the cross-referenced file containing this value.

When you enter a TABLE, TABLEF, or GRAPH request to read a joined data source, use the name of the host file in the opening statement. For example, assume you are writing a report from the JOB and SALARY data sources. You enter the following:

```
JOIN EMP_ID IN JOB TO ALL EMP_ID IN SALARY
```

This command joins the field EMP_ID in the JOB data source to the field EMP_ID in the SALARY data source. The JOB data source is the host file and the SALARY data source is the cross-referenced file. You then enter this TABLE request:

```
TABLE FILE JOB
PRINT SALARY AND JOB_TITLE BY EMP_ID
END
```

The first record retrieved is a JOB file record for employee #071382660. FOCUS then retrieves all records in the SALARY data source containing employee #071382660. This process continues until all the records have been read.

Requests reading joined data sources can contain DEFINEd fields. There are two types of DEFINEd fields:

- Fields defined by the DEFINE command. The DEFINE command must begin with the statement

  ```
  DEFINE FILE hostfile
  ```

  where:

  hostfile

  Is the name of the host file. The DEFINE expressions can contain fields from both the host file and the cross-referenced file.

  Note: Expressions defining the host field are an exception, as explained in *How to Use DEFINE-Based JOIN Syntax* on page 14-6. Since issuing the JOIN command clears all DEFINE commands for the host file and the joined structure, you must enter the DEFINE commands after the JOIN.

- Fields defined by the DEFINE statement in the Master File of the host file.

Syntax How to Create a Simple JOIN

The simple JOIN syntax requires that the two fields you are using to join the files are real fields declared in the Master File.

The syntax of the simple JOIN command is

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

where:

`JOIN field1`

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

`AND field1a ...`

When you are joining two FOCUS files you can specify up to four alphanumeric fields in the host file that, if concatenated, contain values shared with the cross-referenced file. You may not specify more than one field in the cross-referenced file when the suffix of the cross-referenced file is FOC. Interface files do not have this restriction on the cross-referenced file. For example, assume the cross-referenced file contains a date field having a year/month/day format. The host file has a year field, a month field, and a day field. You can specify these three fields to join them to the date field in the cross-referenced file. The JOIN command treats the three fields as one.

You can do a multi-field JOIN or concatenated JOIN with interface products. See the appropriate interface manuals for discussions about JOIN. Up to 16 fields may be used when joining Interface files that support multi-field JOINs.

Note: The keyword AND in the syntax is required. The MODIFY LOOKUP function cannot retrieve data in the cross-referenced segment using the concatenated fields.

`IN host`

Is the name of the host file.

`TAG tag1`

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

Note:

- The SET FIELDNAME command must be set to NEW for field names and aliases that exceed 12 characters.
- The tag name for the host file must be the same in all the JOIN commands of a JOINed structure.

`TO [ALL] field2`

Is the name of a field in the cross-referenced file whose format matches that of *field1* (or of concatenated host fields). This field is called the cross-referenced field. Use the ALL parameter when *field2* may have multiple values in common with one value in *field1*. This is called a non-unique (one-to-many) JOIN. See *Unique and Non-Unique JOIN Structures* on page 14-10.

`AND field2a...`

Field2a is the name of a field in the cross-referenced file with values in common with *field1a*.

Note: *Field2a* may be qualified. This field is only available for interfaces supporting multi-field JOIN.

`IN crfile`

Is the name of the cross-referenced file.

`TAG tag2`

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

Note:

- The SET FIELDNAME command must be set to ON for field names and aliases that exceed 12 characters.
- The tag name for the host file must be the same in all the JOIN commands of a JOINed structure.

`AS joinname`

Is an optional name of up to eight characters that you may assign to the JOIN structure. You must assign a unique name to a JOIN structure if you want to ensure that a subsequent JOIN command will not overwrite it, if you want to clear it selectively later, or if the structure is recursive, as explained in *Recursive JOIN Structures* on page 14-15.

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

`END`

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

Reference Usage Notes for Simple JOIN Syntax

- If tag names are specified in a recursive JOIN, duplicate field names must be qualified with the tag name.
- If a join name is specified and tag names are not specified in a recursive JOIN, duplicate field names must be prefixed with the first four characters of the join name.

- If both a join name and a tag name are specified in a recursive JOIN, the tag name must be used as a qualifier.
- The tag name must be used as the field name qualifier to retrieve duplicate field names in a non-recursive JOIN. The first occurrence of a field name is retrieved without qualification.

Example Creating a Simple Unique JOIN Structure

An example of a simple unique JOIN is shown below:

```
JOIN JOBCODE IN EMPLOYEE TO JOBCODE IN JOBFILE AS JJOIN
```

Syntax How to Use DEFINE-Based JOIN Syntax

DEFINE-based JOIN syntax allows the host field to be created by a DEFINE statement, either in the Master File or in a separate DEFINE command.

If you are creating this field in a DEFINE command, you must issue the DEFINE after the JOIN command but before the TABLE request.

The syntax of the DEFINE-based JOIN command is

```
JOIN deffld WITH dbfld IN hostfile [TAG tag1]
TO [ALL] field2 IN crfile [TAG tag2] [AS joinname]
END
```

where:

`JOIN` *`deffld`*

Is the name of the DEFINEd field for the host file (the host field).

`WITH` *`dbfld`*

Indicates that the host field is a DEFINEd field. *dbfld* is the name of any real field in the host segment associated with the DEFINEd field. To determine which segment contains the DEFINEd field, use the ? DEFINE query.

`IN` *`hostfile`*

Is the name of the host file.

`TAG` *`tag1`*

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

Note:

- The SET FIELDNAME command must be set to ON for field names and aliases that exceed 12 characters.
- The tag name for the host file must be the same in all the JOIN commands of a JOINed structure.

`TO` *`field2`*

Is the name of a data source field in the cross-referenced file whose format matches that of the DEFINEd field. This field must be a real field declared in the Master File.

`ALL`

Is the parameter to use when the cross-referenced field may have multiple values in common with one value in the DEFINEd host field. See *Unique and Non-Unique JOIN Structures* on page 14-10 for more information.

`IN` *`crfile`*

Is the name of the cross-referenced file.

`TAG` *`tag2`*

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

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

`AS` *`joinname`*

Is an optional name of up to eight characters that you may assign to the JOIN structure. You must assign a unique name to a JOIN structure if you want to ensure that a subsequent JOIN command will not overwrite it, if you want to selectively clear it later, or if the structure is recursive, as explained in *Recursive JOIN Structures* on page 14-15.

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

`END`

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

Reference Usage Notes for DEFINE-Based JOIN Syntax

- The DEFINE expression creating the host field may contain only fields in the host file and constants.
- If you are creating the DEFINEd field in a DEFINE command, issue the DEFINE command after the JOIN command is issued, but prior to the TABLE request. A JOIN statement clears all fields created by DEFINE commands for the host file and the joined structure. DEFINEd fields in Master Files are not affected.
- You can also use other DEFINEd fields in your report. (Remember that a TABLE request reading a joined structure can only access DEFINEd fields that are defined either in the host file Master File or in a DEFINE command that names the host file [DEFINE FILE hostfile].) Define these fields for the host file and the joined structure after you issue the JOIN.
- The LOOKUP function in MODIFY requests cannot be used on a DEFINE-based JOIN; DEFINE is not evaluated during a MODIFY.

Example Using a DEFINE-Based JOIN

For example, suppose a retail chain sends four store managers to attend classes. Each person, identified by an ID number, manages a store in a different city. The stores and the cities where they are located are contained in the SALES file. The manager IDs, the classes, and dates the managers attended are contained in the EDUCFILE file.

The following procedure lists the courses that the managers attended, identifying the managers by the cities where they work. Note the three elements in the procedure:

- The JOIN command joins the SALES file to the EDUCFILE, based on the values common to the ID_NUM field (which contains manager IDs) in the SALES file and the EMP_ID field in the EDUCFILE file. Note that the ID_NUM field does not exist yet; it will be created by the DEFINE command below.
- The DEFINE command creates the ID_NUM field, assigning to it the IDs of the managers working in the four cities.
- The TABLE request produces the report.

The procedure is shown below:

```
JOIN ID_NUM WITH CITY IN SALES TO ALL EMP_ID IN EDUCFILE AS SALEDUC

DEFINE FILE SALES
ID_NUM/A9 = DECODE CITY ('NEW YORK' 451123478 'NEWARK' 119265415
                         'STAMFORD' 818692173 'UNIONDALE' 112847612);
END

TABLE FILE SALES
PRINT DATE_ATTEND BY CITY BY COURSE_NAME
END
```

The procedure produces the following report:

```
PAGE 1

CITY           COURSE_NAME                  DATE_ATTEND
----           -----------                  -----------
NEW YORK       FILE DESCRPT & MAINT         81/11/15
NEWARK         BASIC RPT NON-DP MGRS        82/08/24
STAMFORD       BASIC REPORT PREP DP MGRS    82/08/02
               HOST LANGUAGE INTERFACE      82/10/21
UNIONDALE      BASIC REPORT PREP FOR PROG   81/11/16
               FILE DESCRPT & MAINT         81/11/15
```

Joining Fields With Different Datatypes

The JOIN command allows you to join two files containing different numeric datatypes. For example, a short packed field can now be joined to a long packed field, or an integer field joined to a packed field. This provides enormous flexibility for creating reports from joined files.

In order for datatype conversions to take place, FOCUS must access the interface module GNTINT. This allows for conversion of the USAGE in the FROM file to match those of the TO file. When joining to a FOCUS file, it is necessary to issue the command SET JOINOPT = GNTINT.

When joining a shorter field to a longer field, the FROM value is padded to the length of the TO field, adding spaces (for alpha fields) or hexadecimal zeroes (for numeric fields). This new value is used for searches in the cross-referenced file.

When joining a longer field to a shorter field, the FROM value is truncated. If part of your value is truncated due to the length of the USAGE in the cross reference file, only records matching the truncated value will be found in the cross-reference file.

Syntax How to Enable Joins With Datatype Conversion

When joining to a FOCUS file where datatype conversion must take place prior to issuing the JOIN, it is necessary to issue the following command:

SET JOINOPT = GNTINT

Example Issuing Joins With Datatype Conversion

You can join a short packed field to a long packed field. The following two Master Files describe two possible files that can be joined:

```
FILE=PACKED,SUFFIX=FIX,$
  SEGNAME=ONE,SEGTYPE=S0
   FIELD=FIRST,,P8,P4,INDEX=I,$
```

```
FILE=PACKED2,SUFFIX=FIX,$
  SEGNAME=ONE,SEGTYPE=S0
   FIELD=PFIRST,,P31,P16,INDEX=I,$
```

Note: When joining packed fields the preferred sign format of X’C’ for positive values and X’D’ for negative values is still required. All other non-preferred signs are converted to either X’C’ or X’D’.

Unique and Non-Unique JOIN Structures

In a unique JOIN structure, one value in the host field corresponds to one value in the cross-referenced field. In a non-unique JOIN structure, one value in the host field corresponds to multiple values in the cross-referenced field.

The ALL parameter in a JOIN command indicates that the JOIN structure is non-unique.

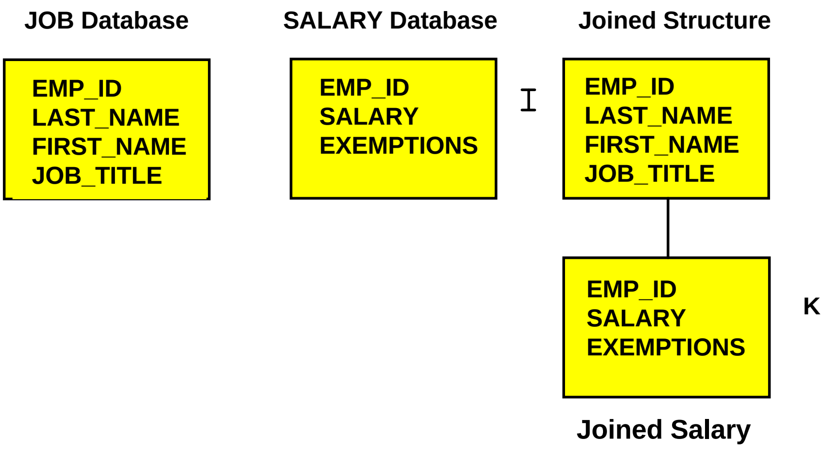
Note:

- Omit the ALL parameter only when you are sure that the JOIN structure is unique. Omitting the ALL parameter reduces overhead.
- The ALL parameter will not interfere with the proper creation of the JOIN structure even if it is unique. Use the ALL parameter if you are not sure whether the JOIN structure is unique. This ensures that your reports will contain all relevant data from the cross-referenced file, regardless of whether the structure is unique.

Example A Unique JOIN Structure

The following example illustrates a unique JOIN structure. Two FOCUS databases are joined together: a JOB database and a SALARY database. Both databases are organized by employee, and they are joined on an EMP_ID (employee ID) field in the root segments of both files. Each employee has one segment instance in the JOB database and one instance in the SALARY database. To join these two databases, issue this JOIN command:

```
JOIN EMP_ID IN JOB TO EMP_ID IN SALARY
```



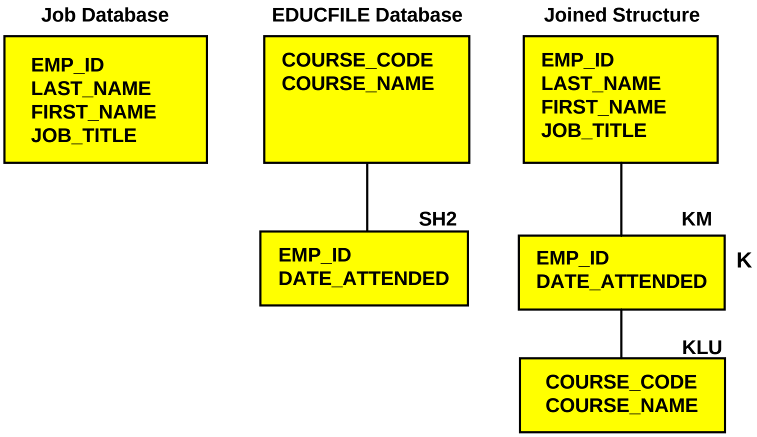
Example A Non-Unique JOIN Structure

If a field value in the host file can appear on many segment instances in the cross-referenced file, then you should include the ALL keyword in the JOIN syntax. This structure is called a non-unique JOIN structure.

For example, assume that two FOCUS databases are joined together: the JOB database and an EDUCFILE database which records employee attendance at in-house courses. The JOIN structure is shown in the diagram below.

The JOB database is organized by employee, but the EDUCFILE database is organized by course. The databases are joined on the EMP_ID field. Since an employee has one position but can attend several courses, the employee has one segment instance in the JOB database but can have many instances in the EDUCFILE database, as many instances as courses attended. To join these two databases, issue the following JOIN command, using the ALL keyword:

```
JOIN EMP_ID IN JOB TO ALL EMP_ID IN EDUCFILE
```



Data Formats of Shared Fields in Host and Cross-Referenced Files

The fields containing the shared values in the host and cross-referenced files must have similar data formats. If you specify multiple host file fields in the JOIN command, the JOIN command treats the fields as one concatenated field. Add the field format lengths to obtain the length of the concatenated field. The rules are listed below:

- If the host field is alphanumeric, the cross-referenced field must also be alphanumeric and have the same length. The formats may have different edit options.
- If the host field is numeric, the host field format, as specified by the USAGE (or FORMAT) attribute in the Master File, must agree in type (I, P, F, or D) with the format of the cross-referenced field as specified by the USAGE (or FORMAT) attribute unless the JOINOPT setting is used.
- The edit options may differ. The length may also differ, but it will have the following effect:

  If the format of the host field (as specified by the USAGE attribute) is packed (P) or integer (I) and is longer than the cross-referenced field format (specified by the USAGE attribute for FOCUS files or the ACTUAL attribute for external files), FOCUS only compares the length of the cross-referenced field format, using only the rightmost digits of the shorter field. For example, a five-digit packed field is joined to a three-digit packed field. When FOCUS retrieves a host record with a five-digit number, it retrieves all cross-referenced records with the last three digits of that number.

  If the format of the host field is double precision (D), FOCUS compares the left-most eight bytes of each field.
- The host and cross-referenced fields can be described as groups in the Master File if they contain the same number of component fields. The corresponding component fields in each group (for example, the first field in the host group and the first field in the cross-referenced group) must obey the above rules.
- If the host field is not a group field, the cross-referenced field can be a group. If the host field is a group, the cross-referenced field must also be a group.

Restrictions on Joining Files

The use of data sources as host files and cross-referenced files in JOINs depends on the types of data sources you are joining. The following data sources can be joined to any other data source in this group and can participate in up to 16 different JOINs: FOCUS, VSAM, ISAM, IMS (DOS/DL1), SQL/DS, DB2, Oracle, Teradata, CA-DATACOM/DB, MODEL 204, ADABAS, CA-IDMS/DB, SUPRA, TOTAL, Millennium, and simple fixed sequential files.

Note:

- Files protected by the FOCUS DBA security feature may be joined from unprotected (non-DBA) files. However, the DBA information is taken from the host file only. See the *Describing Data* manual for details on using DBA with joined structures. When joining files protected by DBA security, the files may possess different DBA passwords.
- You cannot join comma-delimited files.

Restrictions on Cross-Referenced Fields

The cross-referenced field must have the following characteristics:

- In FOCUS databases, the field must be indexed. Indexed fields have the attribute FIELDTYPE=I or Index=I in the Master File. If the cross-referenced field does not have this attribute, append the attribute to the field declaration in the Master File and rebuild the file using the REBUILD utility with the INDEX option. This will add an index to your FOCUS database.
- In CA-DATACOM/DB, SQL/DS, DB2, Oracle, and Teradata tables, the field can be any field.
- In IMS databases, the field must be a key field in the root segment of the database. It can be a primary or secondary index.
- In fixed sequential files, the cross-referenced field can be any field. However, the field must be sorted the same way as it is sorted in the host file, in ascending order. If the data is not in the same sort order, errors will be displayed. If the cross-referenced file contains only one segment, the host file must have a segment declaration.

Restrictions on Group Fields

When group fields are used in a JOIN, the group in the host file and the group in the cross referenced file must have the same number of elements.

- In ISAM files, the field must be the full primary key if you issue a unique JOIN or an initial subset of the primary key if you issue a non-unique JOIN. In the Master File, the primary key is described by a key GROUP; the initial subset is the first field in that group.
- In VSAM KSDS files, the field must be the full primary or alternate key if you issue a unique JOIN or an initial subset of the primary or alternate key if you issue a non-unique JOIN. In the Master File, the primary key is described by a key GROUP; the initial subset is the first field in that group.

  In VSAM ESDS files, the field can be any field, as long as the file is already sorted on that field.
- In Model 204 databases, the field must be a key field. In the Access File, the types of key fields are alphanumeric (KEY), ordered character (ORA), ordered numeric (ORN), numeric range (RNG), invisible (IVK), and invisible range (IVR).
- In ADABAS files, the field must be a descriptor field, a superdescriptor defined with the .SPR or .NOP field name suffix, or a subdescriptor defined with the .NOP field name suffix. The field description in the Master File must contain the attribute FIELDTYPE=I.

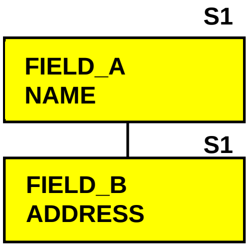
  In the Access File, the cross-referenced segment must specify ACCESS=ADBS and either CALLTYPE=FIND or CALLTYPE=RL. If CALLTYPE=RL, the host field can be JOINed to the high-order portion of a descriptor, superdescriptor, or subdescriptor, if the high-order portion is longer than the host field.
- In CA-IDMS/DB files, the field must be an indexed field on a network record, identified by the attribute FIELDTYPE=I in the Master File; a CA-IDMS/DB CALC field on a network record, identified by the CLCFLD keyword in the Access File; or any field on an LRF or ASF record.

Recursive JOIN Structures

You can join a FOCUS or IMS file to itself, creating a recursive structure. In the most common type of recursive structure, a parent segment is joined to a descendant segment, so that the parent becomes the child of the descendant. This technique (useful for databases storing bills of materials, for example) enables you to report from data sources as if they have more levels of segments than they really do.

Example Understanding Recursive JOIN Structures

For example, the GENERIC data source shown below consists of Segments A and B.

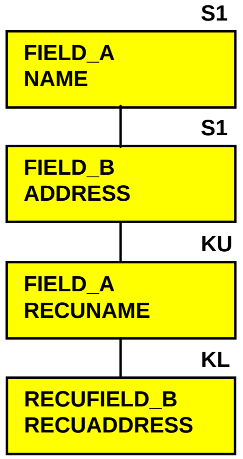


GENERIC Data Source

To create a recursive structure, enter the following:

```
JOIN FIELD_B IN GENERIC TO FIELD_A IN GENERIC AS RECURSIV
```

This results in the joined structure (shown below).



GENERIC Data Source
Recursively Joined

Note that the two segments are repeated on the bottom. To refer to the fields in the repeated segments (other than the field to which you are joining), append the first four letters of the JOIN name to the field names and aliases. In the above example, the JOIN name is RECURSIV. Thus you should refer to FIELD_B in the bottom segment as RECUFIELD_B or use tags.

Reference Usage Notes for Recursive JOIN Structures

- You must either specify a unique JOIN name or use tag names in the JOIN command. Otherwise, you will not be able to refer to the fields in the repeat segments at the bottom of the JOIN structure.
- You may use a DEFINE-based JOIN (see *How to Use DEFINE-Based JOIN Syntax* on page 14-6) to join a DEFINEd field in a descendant segment to a field in the parent segment.
- You can extend a recursive structure still further by issuing as many JOINs that join the bottom repeat segment in the structure to the parent segment as there are up to 16 levels.
- For FOCUS files, the field in the parent segment to which you are joining must be indexed.
- For IMS files, the following applies:
  - The parent segment must be the root segment of the database.
  - The field to which you are joining must be both a key field and a primary or secondary index.
  - You need a duplicate PCB in the PSB for every recursive JOIN you create.

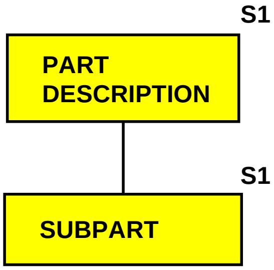
Example Using Recursive JOIN Structures

This example explains how to use recursive joins to store and report on a bill of materials. Suppose you are designing a data source called AIRCRAFT that contains the bill of materials for a company’s aircraft. The data source records data on three levels of airplane parts:

- Major divisions, such as the cockpit or cabin.
- Parts of divisions, such as instrument panels and seats.
- Subparts, such as nuts and bolts.

The data source must record each part, the part description, and the smaller parts composing the part. Some parts, such as nuts and bolts, are common throughout the aircraft. If you design a three-segment database, one segment for each level of parts, the database will repeat descriptions of common parts every place they are used.

To reduce this repetition, design a data source that has only two segments (shown in the following diagram). The top segment describes each part of the aircraft, large and small. The bottom segment lists the component parts without descriptions.

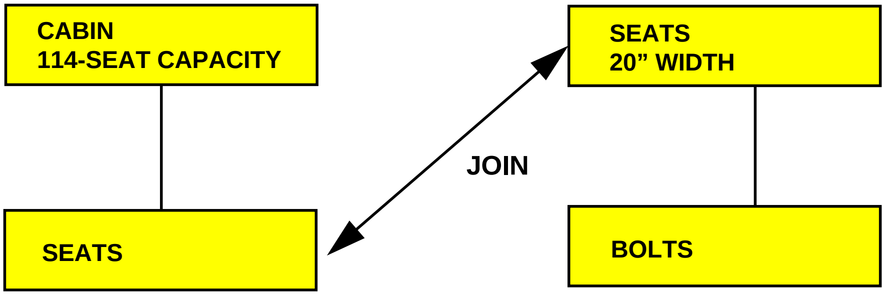


AIRCRAFT Data Source

Every part (except the largest divisions) appears in both the top segment, where it is described, and in the bottom segment, where it is listed as one of the components of a larger part. (The smallest parts, such as nuts and bolts, appear in the top segment without an instance of a child in the bottom segment.) Note that each part, no matter how often it is used in the aircraft, is described only once.

If you join the bottom segment to the top segment, FOCUS can retrieve the descriptions of component parts in the bottom segment. It can also relate the first-level major divisions to third-level small parts, going from the first-level to the second-level to the third-level. Thus, the structure behaves as a three-level data source although it is actually a more efficient two-level.

For example, CABIN is a first-level division appearing in the top segment. It lists SEATS as a component in the bottom segment. SEATS also appears in the top segment; it lists BOLTS as a component in the bottom segment. If you join the bottom segment to the top segment, you will enable FOCUS to go from CABIN to SEATS and from SEATS to BOLTS.

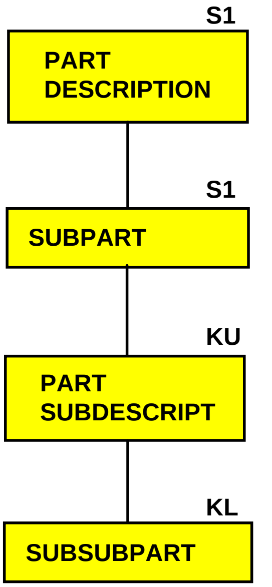


Sample Instance in the AIRCRAFT Data Source

Join the bottom segment to the top segment with this JOIN command:

```
JOIN SUBPART IN AIRCRAFT TO PART IN AIRCRAFT AS SUB
```

This creates the following recursive structure.



AIRCRAFT Data Source Recursively Joined

You can then produce a report on all three levels of data with this TABLE command (the field SUBDESCRIPT describes the contents of the field SUBPART):

```
TABLE FILE AIRCRAFT
PRINT SUBSUBPART BY PART BY SUBPART BY SUBDESCRIPT
END
```

JOIN Structure Considerations

When you join two data sources together, FOCUS treats the data sources as one logical structure. This structure results from appending the structure of the cross-referenced file to the structure of the host file. The segment in the cross-referenced file containing the shared value field becomes the child of the segment in the host file with the shared value field. To display the JOIN structure, issue this command in a stored procedure

```
CHECK FILE hostfile PICTURE
```

where:

hostfile

Is the name of the host file. A sample structure follows.

The segments belonging to the host file appear as regular segments outlined by asterisks; the segments belonging to the cross-referenced file appear as virtual segments outlined by dots. The segments of the cross-referenced file are also labeled with the cross-referenced file name below each segment.

```
JOIN EMP_ID IN JOB TO EMP_ID IN SALARY
>
CHECK FILE JOB PICTURE
 NUMBER OF ERRORS=     0
 NUMBER OF SEGMENTS=   2  ( REAL=    1  VIRTUAL=   1 )
 NUMBER OF FIELDS=     7  INDEXES=   0  FILES=     2
 TOTAL LENGTH OF ALL FIELDS=   86
SECTION 01
             STRUCTURE OF FOCUS    FILE JOB      ON 06/26/96 AT 15.09.03

         JOBSEG
 01      S1
**************
*EMP_ID      **
*FIRST_NAME  **
*LAST_NAME   **
*JOB_TITLE   **
*            **
***************
 **************
       I
       I
       I
       I SALSEG
 02    I KU
..............
:EMP_ID      :K
:SALARY      :
:EXEMPTIONS  :
:            :
:            :
:............:
 JOINED  SALARY
```

The top segment of the cross-referenced file structure is the one containing the shared-value field. If this segment is not the root segment, the cross-referenced file structure is inverted as it would be in an alternate file view.

The cross-referenced file segment types in the JOIN structure are the following:

- In unique JOIN structures, the top cross-referenced file segment has the segment type KU. Its unique child segments have segment type KLU; non-unique child segments have segment type KL.
- In non-unique JOIN structures, the top cross-referenced file segment has the segment type KM. Its unique child segments have segment type KLU; non-unique child segments have segment type KL.

The host file structure remains unchanged. The cross-referenced file may still be used independently.

Listing JOIN Structures: The ? JOIN Query

To display a list of joined data sources, issue the following command at the FOCUS command line or in a stored procedure:

```
? JOIN
```

This displays every JOIN command currently in effect. For example:

```
 JOINS CURRENTLY ACTIVE

HOST                     CROSSREFERENCE
FIELD      FILE   TAG    FIELD      FILE     TAG    AS         ALL
-----      ----   ---    -----      ----     ---    --         ---
EMP_NUM    JOB           EMP_ID     SALARY          JOBSAL
```

If the JOIN structure has no joinname, the AS phrase is omitted. If two data sources are joined by multiple JOIN commands, only the first command you issued is displayed.

Clearing JOIN Structures: The JOIN CLEAR Command

To clear a JOIN structure, issue this command at the FOCUS command line or in a stored procedure

```
JOIN CLEAR {joinname|*}
```

where:

joinname

Is the AS name of the JOIN structure you want to clear.

*

Clears all JOIN structures.

15 Merging Data Files

Topics:

- Merging Data Files: MATCH
- Universal Concatenation
- Merging Concatenated Data Sources With MATCH and MORE
- Cartesian Product

FOCUS provides a variety of features for customizing tabular reports. This chapter describes how to gather data for your reports by merging the contents of data structures with the MATCH command or concatenating databases with the MORE command, and reporting from the combined data.

Merging Data Files: MATCH

You can merge two or more data files, and specify which records will be merged and which will be screened out, using the MATCH command. The command creates a new data file—a HOLD file—into which it merges fields from the selected records. You can report from the new data file and use it as you would any other HOLD file (HOLD files are described in Chapter 12, *Saving and Reusing Your Report Output)*. The merge process does not change the original data files.

You select which records are merged into the new data file and which records are excluded by specifying sort fields in the MATCH command. You specify one set of sort fields—using the BY phrase—for the first data file, and a second set of sort fields for the second data file. The MATCH command compares all sort fields that have been specified in common for both data files, and then merges into the new HOLD file all records from the first data file whose sort values match those in the second data file.

In addition to merging data file records that share values, you can merge records based on other relationships. For example, you can merge all records whose values do not match—that is, all records in each data file whose sort values are not matched in the other data file. Yet another type of merge combines all records from the first data file with any matching records from the second data file.

You can merge up to six sets of data in one MATCH request—for example, merging six different data files, or merging data from the same data file more than once.

Syntax How to Merge Data Files

The syntax of the MATCH command is similar to that of the TABLE command:

```
MATCH FILE filename
.
.
.
RUN
FILE filename
.
.
.
[AFTER MATCH merge_phrase]
RUN
FILE filename
.
.
.
[AFTER MATCH merge_phrase]
END
```

A RUN command must follow each AFTER MATCH phrase (except for the last one). The END command must follow the final AFTER MATCH phrase.

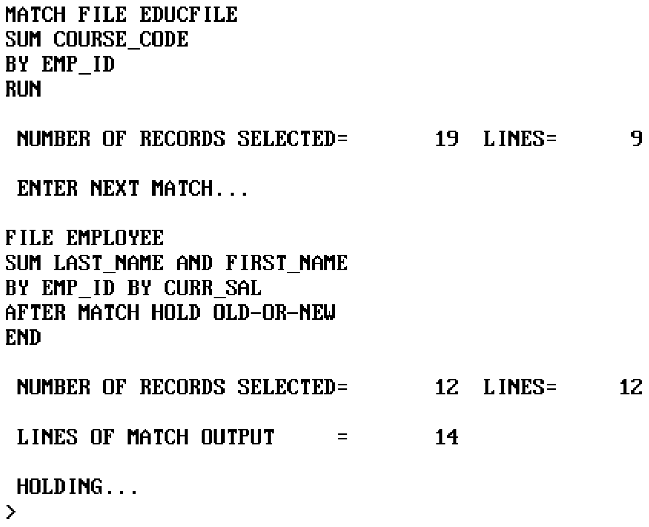
MATCH generates a single-segment HOLD file. You can print the contents of the HOLD file using the PRINT command with the wildcard character (*).

Reference Usage Notes for Merging Data Files

- The ACROSS and WHERE TOTAL phrases, and the COMPUTE command, are not permitted in a MATCH request. You can, however, use the DEFINE command.
- A total of 32 BY phrases and 256 display fields can be used in each MATCH request.
- You must specify at least one BY field for each file used in the MATCH request.
- When used with MATCH, the SET HOLDLIST command behaves as if HOLDLIST were set to ALL.
- You cannot use BY HIGHEST in a MATCH request.

Example Merging Data Files

After you enter the keyword RUN, FOCUS indicates how many records were retrieved and — if you are entering the MATCH request at the command line—prompts you for the name of the next data file to be merged:

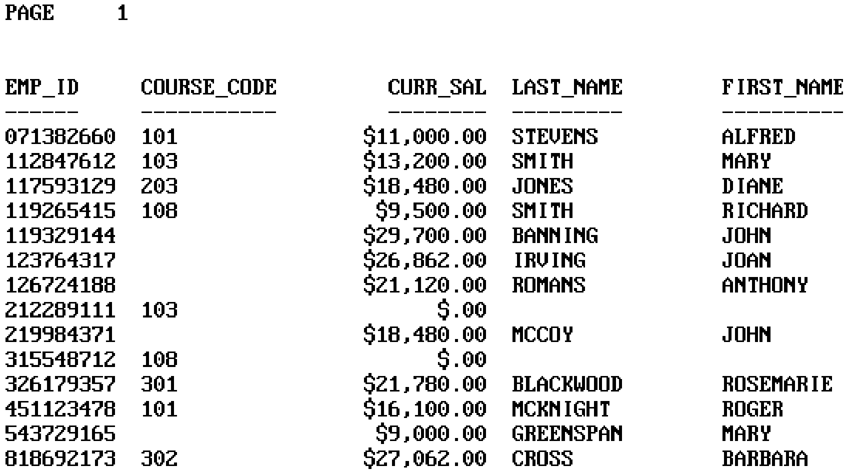


```
MATCH FILE EDUCFILE
SUM COURSE_CODE
BY EMP_ID
RUN

 NUMBER OF RECORDS SELECTED=       19  LINES=      9

 ENTER NEXT MATCH...

FILE EMPLOYEE
SUM LAST_NAME AND FIRST_NAME
BY EMP_ID BY CURR_SAL
AFTER MATCH HOLD OLD-OR-NEW
END

 NUMBER OF RECORDS SELECTED=       12  LINES=     12

 LINES OF MATCH OUTPUT     =       14

 HOLDING...
>
```

The merge phrase used in this example was OLD-OR-NEW. This means that records from both the first (old) data file plus the records from the second (new) data file will appear in the HOLD file:

```
-*****************************
-*  PRINT CONTENTS OF HOLD FILE
-*****************************
TABLE FILE HOLD
PRINT *
END
```

This request produces the following report.



```
PAGE     1


EMP_ID     COURSE_CODE         CURR_SAL  LAST_NAME        FIRST_NAME
------     -----------         --------  ---------        ----------
071382660  101               $11,000.00  STEVENS          ALFRED
112847612  103               $13,200.00  SMITH            MARY
117593129  203               $18,480.00  JONES            DIANE
119265415  108                $9,500.00  SMITH            RICHARD
119329144                    $29,700.00  BANNING          JOHN
123764317                    $26,862.00  IRVING           JOAN
126724188                    $21,120.00  ROMANS           ANTHONY
212289111  103                     $.00
219984371                    $18,480.00  MCCOY            JOHN
315548712  108                     $.00
326179357  301               $21,780.00  BLACKWOOD        ROSEMARIE
451123478  101               $16,100.00  MCKNIGHT         ROGER
543729165                     $9,000.00  GREENSPAN        MARY
818692173  302               $27,062.00  CROSS            BARBARA
```

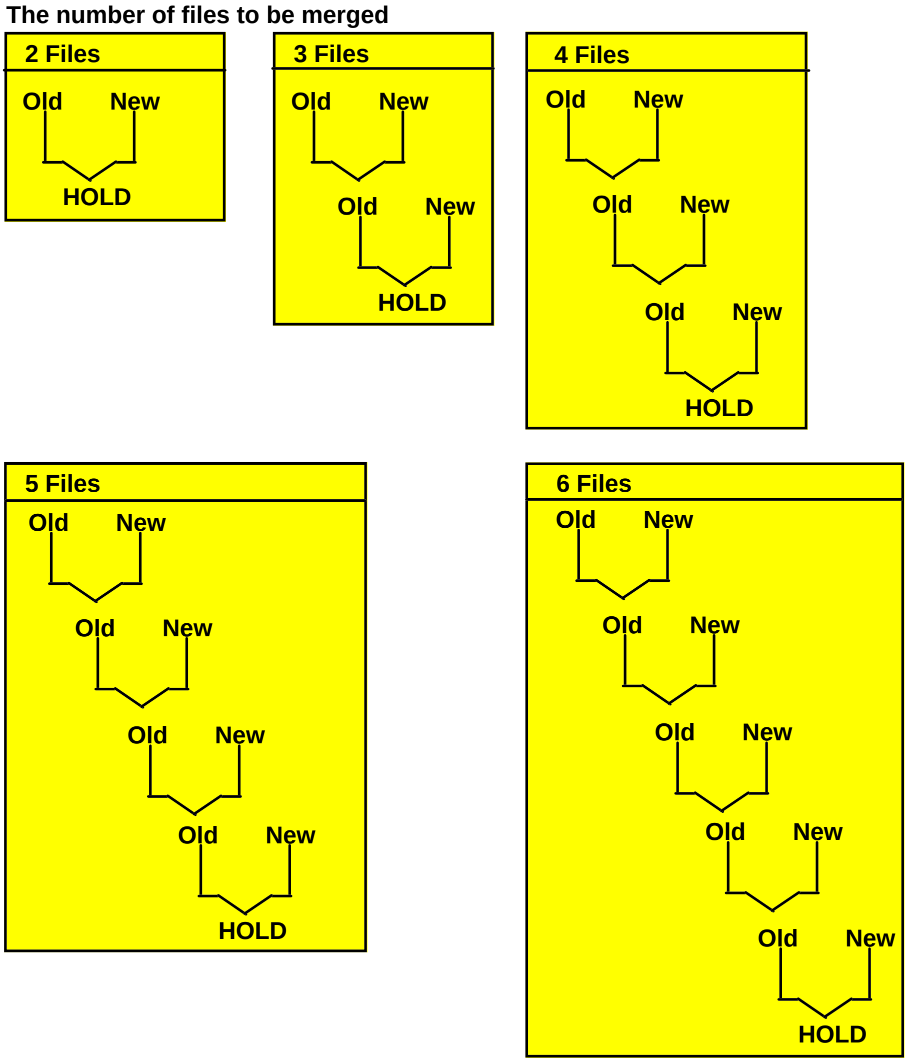
MATCH Processing

The actual way MATCH merges data depends on the order in which you name data files in the request, the BY fields, display commands, and the merge phrases you use. In general, however, processing is as follows:

1. MATCH retrieves requested records from the first data file you name and writes them to a temporary work area.
2. MATCH retrieves requested records from the second data file you name and writes them to a temporary work area.
3. It compares the retrieved records’ common high-order sort fields as specified in the merge phrase (for example, OLD-OR-NEW).
4. It writes the merged results of the comparison to a temporary data file (if there are more MATCH operations). It cycles through all data files named until END is encountered.
5. It writes final records to the HOLD file.

Merge Phrases

MATCH logic depends on the concept of old and new data files. Old refers to the first data file named in the request and new refers to the second data file. The result of each merge creates a HOLD file until the END command is encountered. The following diagram illustrates the general merge process:



Syntax How to Specify Merge Phrases

The syntax is

```
AFTER MATCH HOLD [AS 'name'] mergetype
```

where:

`AS 'name'`

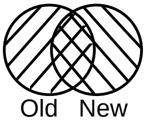
Specifies the name of the extract data file created by the MATCH command. The default is HOLD.

`mergetype`

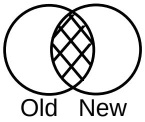
Specifies how the retrieved records from the files are to be compared.

The results of each phrase are graphically represented using Venn diagrams. In the diagrams, the left circle represents the old data file, the right circle represents the new data file, and the shaded areas represent the data that is written to the HOLD file.

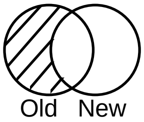
`OLD-OR-NEW` specifies that all records from both the old data file and the new data file will appear in the HOLD file. This is the default if AFTER MATCH line is omitted.



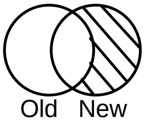
`OLD-AND-NEW` specifies that records that appear in both the old and new data files appear in the HOLD file. (The intersection of the sets.)



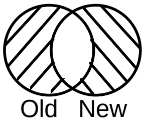
`OLD-NOT-NEW` specifies that records that appear only in the old data file will appear in the HOLD file.



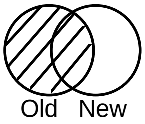
`NEW-NOT-OLD` specifies that records that appear only in the new data file will appear in the HOLD file.



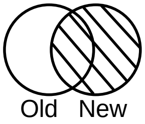
`OLD-NOR-NEW` specifies that only records which are in the old data file but not in the new data file, or in the new data file but not in the old, will appear in the HOLD file (the complete set of non-matching records from both data files).



`OLD` specifies that all records from the old data file, and any matching records from the new data file, will be merged into the HOLD file.



`NEW` specifies that all records from the new data file, and any matching records from the old data file, will be merged into the HOLD file.



BY Fields as Common High Order Sort Fields

When you construct your MATCH so that the first sort field (called the common high-order sort field) used for both data files is the same, FOCUS performs the match by comparing the values of the common high-order sort fields. If the entire sequence of sort fields is common to both files, all are compared.

At least one pair of sort fields is required; field formats must be the same. In some cases, you can redefine a field’s format using the DEFINE command. If the field names differ, use the AS option to rename the second sort field to match the first. When the AS option is used in a MATCH request, the specified field is automatically renamed in the resulting HOLD file.

Example Using a Common High Order Sort Field

For example, this request combines data from databases JOBFILE and PROD. The sort fields are JOBCODE and PROD_CODE, renamed as JOBCODE:

```
MATCH FILE JOBFILE
PRINT JOB_DESC
BY JOBCODE
RUN
FILE PROD
PRINT PROD_NAME
BY PROD_CODE AS 'JOBCODE'
AFTER MATCH HOLD OLD-OR-NEW
END
```

MATCH Processing With Common High Order Sort Fields

To understand common high order sort fields more clearly, consider some of the data from the following data files,

EMPLOYEE Database		**EDUCFILE Database**	
EMP_ID	LAST_NAME	EMP_ID	COURSE_CODE
071382660	STEVENS	071382660	101
119329144	BANNING	21228911	103
112847612	SMITH	112847612	103

and this MATCH request:

```
MATCH FILE EMPLOYEE
SUM LAST_NAME BY EMP_ID
RUN
FILE EDUCFILE
SUM COURSE_CODE BY EMP_ID
AFTER MATCH HOLD OLD-OR-NEW
END
```

MATCH processing is as follows:

- Since there is a common high-order sort field (EMP_ID), FOCUS begins the MATCH logic by matching the EMP_ID values in the first records of EMPLOYEE and EDUCFILE.
- The first records match (each has the same EMP_ID), so FOCUS writes Record 1 to the HOLD file:

  ```
  Record 1: 071382660 STEVENS 101
  ```

- The second records match (each has the value 112847612), so FOCUS writes Record 2:

  ```
  Record 2: 112847612 SMITH 103
  ```

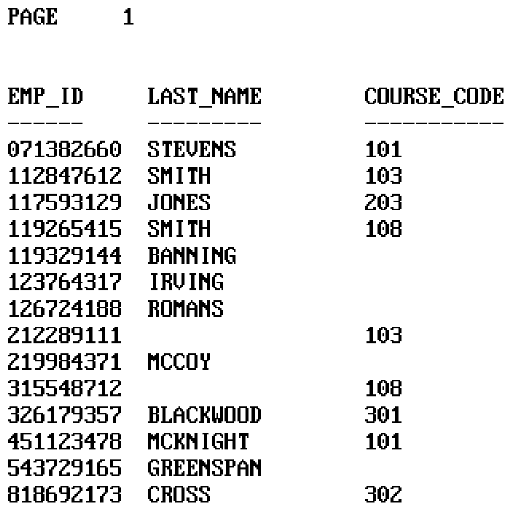
- When the fifth records do not match (EMPLOYEE has the value 119329144 and EDUCFILE has the value 212289111), FOCUS writes the record with the lower value, and inserts a space for the missing value:

  ```
  Record 5: 119329144 BANNING
  ```

- Similarly, the 21228911 record exists only in EMPLOYEE, and is written as:

  ```
  Record 8: 21228911 103
  ```

The complete HOLD file looks like this:



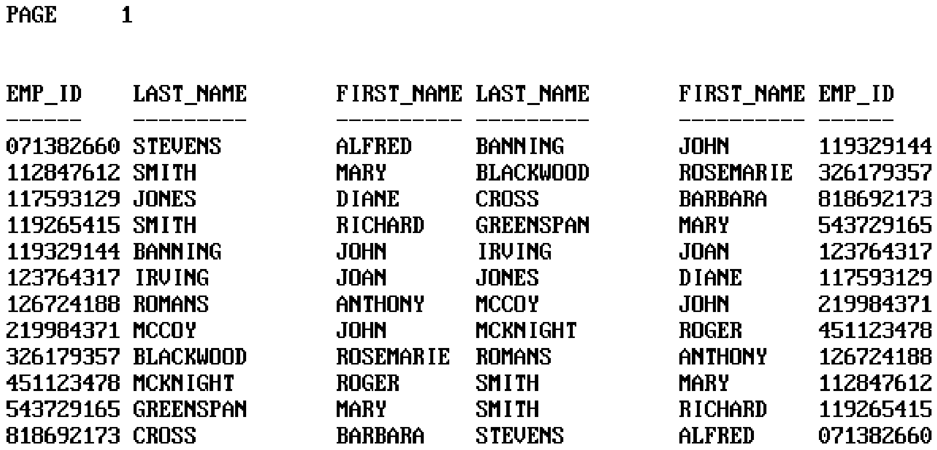
```
PAGE     1


EMP_ID     LAST_NAME          COURSE_CODE
------     ---------          -----------
071382660  STEVENS            101
112847612  SMITH              103
117593129  JONES              203
119265415  SMITH              108
119329144  BANNING
123764317  IRVING
126724188  ROMANS
212289111                     103
219984371  MCCOY
315548712                     108
326179357  BLACKWOOD          301
451123478  MCKNIGHT           101
543729165  GREENSPAN
818692173  CROSS              302
```

Example Merging Without a Common High Order Sort Field

If there are no common high-order sort fields, a match is performed on a record-by-record basis. For example,

```
MATCH FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY EMP_ID
RUN
FILE EMPLOYEE
PRINT EMP_ID
BY LAST_NAME BY FIRST_NAME
AFTER MATCH HOLD OLD-OR-NEW
END

TABLE FILE HOLD
PRINT *
END
```

produces the HOLD file. FOCUS has written the records it finds in the two data files; it does not compare any values. The report is as follows:



```
PAGE     1


EMP_ID    LAST_NAME       FIRST_NAME LAST_NAME       FIRST_NAME EMP_ID
------    ---------       ---------- ---------       ---------- ------
071382660 STEVENS         ALFRED     BANNING         JOHN       119329144
112847612 SMITH           MARY       BLACKWOOD       ROSEMARIE  326179357
117593129 JONES           DIANE      CROSS           BARBARA    818692173
119265415 SMITH           RICHARD    GREENSPAN       MARY       543729165
119329144 BANNING         JOHN       IRVING          JOAN       123764317
123764317 IRVING          JOAN       JONES           DIANE      117593129
126724188 ROMANS          ANTHONY    MCCOY           JOHN       219984371
219984371 MCCOY           JOHN       MCKNIGHT        ROGER      451123478
326179357 BLACKWOOD       ROSEMARIE  ROMANS          ANTHONY    126724188
451123478 MCKNIGHT        ROGER      SMITH           MARY       112847612
543729165 GREENSPAN       MARY       SMITH           RICHARD    119265415
818692173 CROSS           BARBARA    STEVENS         ALFRED     071382660
```

Display Commands and Merging Data Files

The use of PRINT and SUM can fine-tune the MATCH process. To understand their difference, you should have an understanding of the one-to-many relationship: SUM generates one record from many, while PRINT prints each individual record. Through proper choices of BY fields, it is possible to use only the SUM command and get the same result as using PRINT.

Example Using Display Commands in MATCH Processing

To best illustrate the effects of PRINT and SUM on the MATCH process, consider data files A and B and the following requests:

```
     A               B

F1  F2  F3      F1  F4  F5

1   x   100     1   a   10
2   y   200     1   b   20
                2   c   30
                2   d   40
```

This request sums the fields F2 and F3 from file A, sums the fields F4 and F5 from file B, and uses F1 as the common high order sort field.

```
MATCH FILE A
SUM F2 AND F3 BY F1
RUN
FILE B
SUM F4 AND F5 BY F1
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains the following data:

```
F1   F2   F3   F4   F5

1    x    100  b    30
2    y    200  d    70
```

Note that the resulting file contains only 1 record for each common high order sort field.

This request sums fields F2 and F3 from file A, prints fields F4 and F5 from file B, and uses F1 as the common high order sort field.

```
MATCH FILE A
SUM F2 AND F3 BY F1
RUN
FILE B PRINT F4 AND F5 BY F1
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains:

```
F1   F2   F3   F4   F5

1    x    100  a    10
1    x    100  b    20
2    y    200  c    30
2    y    200  d    40
```

Note that the records from file A are duplicated for each record from file B.

This request prints fields F2 and F3 from file A, sums fields F4 and F5 from file B, and uses F1 as the common high order sort field.

```
MATCH FILE A
PRINT F2 AND F3 BY F1
RUN
FILE B
SUM F4 AND F5 BY F1
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains:

```
F1   F2   F3   F4   F5

1    x    100  b    30
2    y    200  d    70
```

Note that each record from file A is included, but only the last record from file B for each common high order sort field.

This request prints fields F2 and F3 from file A, prints fields F4 and F5 from file B, and uses F1 as the common high order sort field.

```
MATCH FILE A
PRINT F2 AND F3 BY F1
RUN
FILE B PRINT F4 AND F5 BY F1
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains:

```
F1   F2   F3   F4   F5

1    x    100  a    10
1         0    b    20
2    y    200  c    30
2         0    d    40
```

Note the blank value for F2 and the 0 for F3.

This request sums the fields F2 and F3 from file A, sums the field F5 from file B and sorts it by field F1, the common high order sort field, and by F4.

```
MATCH FILE A
SUM F2 AND F3 BY F1
RUN
FILE B
SUM F5 BY F1 BY F4
AFTER MATCH HOLD OLD-OR-NEW
END
```

The HOLD file contains:

```
F1   F2   F3   F4   F5

1    x    100  a    10
1    x    100  b    20
2    y    200  c    30
2    y    200  d    40
```

Note that the records for file A are printed for every occurrence of the record in file B.

Universal Concatenation

With Universal Concatenation, you can retrieve data from unlike data source types in a single request; all data, regardless of source, appears to come from a single file. The FOCUS command, MORE, can concatenate all types of data sources (such as, FOCUS, DB2, IMS, VSAM), provided they share corresponding fields with the same format. You can use IF and WHERE selection tests in conjunction with MORE.

To use MORE, you must divide your request into:

- One main request that retrieves the first file and defines the data fields, sorting criteria, and output format for all data.
- Subrequests that define the files and fields to be concatenated to the data of the main request. The fields printed and sorted by the main request must exist in each concatenated file. If they do not, you must create them as DEFINE fields.

During retrieval, FOCUS gathers data from each data source in turn. It then sorts all data and formats the output as described in the main request.

Syntax How to Concatenate Data Sources

The MORE command, accessible within the TABLE and MATCH commands, specifies how to concatenate data from files with dissimilar Master Files.

The syntax for a request that concatenates data sources is

```
{TABLE|MATCH}  FILE file1

  main request
MORE
FILE file2
  subrequest
MORE
FILE file3
  subrequest
MORE
   .
   .
   .
{END|RUN}
```

where:

`TABLE|MATCH`

Begins the request that concatenates the data sources.

`file1`

Is the name of the first file.

`main request`

Is a request, without END or RUN, that describes the sorting, formatting, aggregation, and COMPUTE field definitions for all data. IF and WHERE phrases in the main request apply only to *file1*.

`MORE`

Begins a subrequest. There is no limit to the number of subrequests (other than available memory).

`FILE` *`file2`*

Defines *file2* as the second file for concatenation.

`subrequest`

Is a subrequest. Subrequests can only include WHERE and IF phrases.

`END|RUN`

Ends the request.

Example Concatenating Data Sources

For example, both the EMPLOYEE file and the EXPERSON file contain employee information. You can concatenate their common data into a single file:

- The EMPLOYEE file contains the field values EMP_ID=123456789 and CURR_SAL=50.00.
- The EXPERSON file contains the field values SSN=987654321 and WAGE=100.00.

The following annotated request concatenates the two files:

```
    DEFINE FILE EXPERSON
1.  EMP_ID/A9 = SSN;
    CURR_SAL/D12.2 = WAGE;
    END
2.  TABLE FILE EMPLOYEE
       PRINT CURR_SAL
       BY EMP_ID
3.  MORE
    FILE EXPERSON
    END
```

1. The request must re-map the field names and formats in the EXPERSON file to match those used in the main request.
2. The main request names the first file in the concatenation, EMPLOYEE. It also defines the print and sort fields for both files.
3. The MORE command starts the subrequest that concatenates the next file, EXPERSON. No display commands may be coded in the subrequest. IF and WHERE tests are permitted and are the only syntax that can be coded in a subrequest.

Field Name and Format Matching

All fields referenced in the main request must either exist with the same names and formats in all the concatenated files, or be re-mapped to those names and formats via DEFINE fields. Referenced fields include those used in COMPUTE commands, headings, aggregation phrases, sort phrases, and the PRINT, LIST, SUM, COUNT, WRITE, or ADD commands.

A successful format match means that:

Usage Format Type	**Correspondence**
A	Format type and length must be equal.
I, F, D	Format type must be the same.
P	Format type and scale must be equal.
DATE (new)	Always correspond.
DATE (old)	Edit options must be the same.

Note: Text (TX) fields cannot be concatenated.

Example Matching Field Names and Formats

The following annotated example concatenates data from the EMPDATA and PAYHIST files. Appendix A, *Master Files and Diagrams*, contains the Master Files referenced in the request.

```
    DEFINE FILE EMPDATA
1.  NEWID/A11 = EDIT (ID,'999-99-9999');
    END

    DEFINE FILE PAYHIST
1.  NEWID/A11 = EDIT (SSN,'999-99-9999');
    CSAL/D12.2M = NEW_SAL;
    END

2.  TABLE FILE EMPDATA
    HEADING
    "EMPLOYEE SALARIES"
    " "
3.  PRINT CSAL
3.  BY NEWID AS 'EMPLOYEE ID'
4.  WHERE CSAL GT 65000

5.  MORE
    FILE PAYHIST
6.  WHERE NEW_SAL GT 500
    END
```

1. Defines NEWID with the same name and format as the sort field referenced in the main request.
2. The main request contains all formatting for the resulting report and names the first file to be concatenated.
3. The main request also contains all printing and sorting information. The fields printed and the sort fields must exist as real or DEFINE fields in each file.
4. The WHERE phrase in the main request applies only to the EMPDATA file.
5. The MORE command concatenates the PAYHIST file to EMPDATA.
6. This WHERE phrase applies only to the PAYHIST file. Notice that it references a field that is not defined in the EMPDATA file.

In the resulting report, the EMPLOYEE ID values that start with 000 are from the EMPDATA file, and the values that start with 100 are from the PAYHIST file:

```
PAGE      1

  EMPLOYEE SALARIES

  EMPLOYEE ID             SALARY
  -----------             ------

  000-00-0030         $70,000.00
  000-00-0070         $83,000.00
  000-00-0200        $115,000.00
  000-00-0230         $80,500.00
  000-00-0300         $79,000.00
  100-10-1689            $842.90
                         $982.90
  100-11-9950            $508.75
  100-14-2166            $876.45
  100-15-5843            $508.75
  100-16-2791            $567.89
  100-16-4984          $1,236.78
  100-17-5025            $734.56
  100-18-9299            $567.89
```

Merging Concatenated Data Sources With MATCH and MORE

You can use the MORE command in a MATCH request to merge up to six sets of concatenated files.

You must meet all MATCH requirements (discussed in *Merging Data Files: MATCH* on page 15-1) in the main request. All files to be merged must be sorted by at least one field with a common format.

The MATCH request results in a HOLD file containing the merged data. You can specify how you want each successive file merged using an AFTER MATCH phrase. For example, you can retain:

- All records from both files (OLD-OR-NEW). This is the default.
- Only records common to both files (OLD-AND-NEW).
- Records from the first file with no match in the second file (OLD-NOT-NEW).
- Records from the second file with no match in the first file (NEW-NOT- OLD).
- All non-matching records from both files; that is, records that were in either one of the files but not both (OLD-NOR-NEW).
- All records from the first file with all matching records from the second file (OLD).
- All records from the second file with all matching records from the first file (NEW).

Syntax

How to Merge Concatenated Data Sources

The syntax for a MATCH request against concatenated files is:

```
1.  MATCH FILE file1
        main request
    MORE
2.  FILE file2
        subrequest
    MORE
3.  FILE file3
        subrequest
    RUN
4.  FILE file4
        main request
5.  [AFTER MATCH merge_phrase]
    MORE
6.  FILE file5
        subrequest
    MORE
7.  FILE file6
        subrequest
    RUN
8.  FILE file7
        main request
9.  [AFTER MATCH merge_phrase]
    MORE
10. FILE file8
        subrequest
    MORE
11. FILE file9
        subrequest
    END
```

1. Starts the first answer set in the MATCH. *File1* is the first file in the first answer set.

2. Concatenates *file2* to *file1* in the first MATCH answer set.

3. Concatenates *file3* to *file1* and *file2* in the first MATCH answer set.

4. Starts the second answer set in the MATCH. *File4* is the first file in the second answer set.

5. All data concatenated in the first answer set is merged with the data concatenated in the second answer set using the AFTER MATCH merge_phrase in the second answer set.

6. Concatenates *file5* to *file4* in the second MATCH answer set.

7. Concatenates *file6* to *file4* and *file5* in the second MATCH answer set.

8. Starts the third answer set in the MATCH. *File7* is the first file in the third answer set.

9. All merged data from the first and second answer sets, now a HOLD file, is merged with the data concatenated in the third answer set using the AFTER MATCH merge_phrase in the third answer set. This final set of merged data is stored in a HOLD file.

10. Concatenates *file8* to *file7* in the third MATCH answer set.
11. Concatenates *file9* to *file7* and *file8* in the third MATCH answer set.

Using Sort Fields

If the files in the MATCH share common high-order sort fields with identical names and formats, FOCUS performs the MATCH by merging records with matching sort field values from each of the files. If the two files in the MATCH have the same sort field with different names, you can change one of the names with an AS phrase.

If the files in the MATCH do not share a high-order sort field, FOCUS does not compare fields; it merges the fields from the first record in each file to create the first record in the HOLD file, and so on for all remaining records.

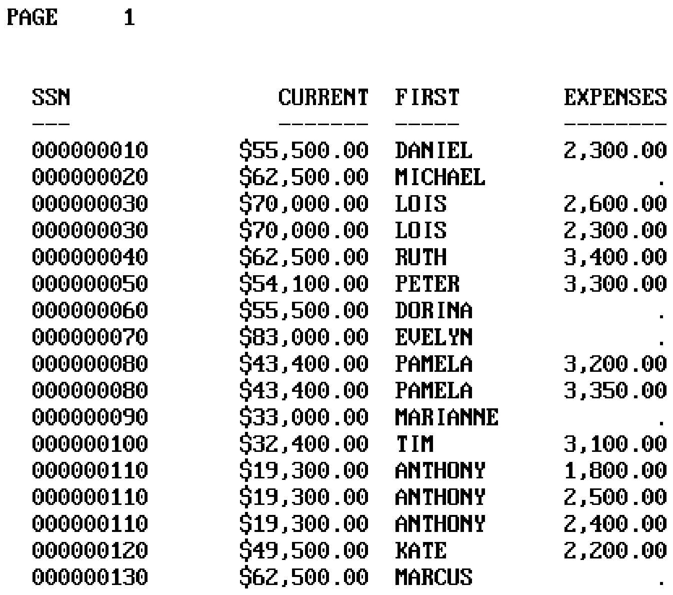
Example Merging Concatenated Data Sources With Common High Order Sort Fields

The following annotated sample stored procedure illustrates MATCH with MORE using a common sort field:

```
1.  DEFINE FILE EMPDATA
    CURR_SAL/D12.2M = CSAL;
    FIRST_NAME/A10 = FN;
    EID/A9 = PIN;
    END

    -*Start MATCH.

2.  MATCH FILE EMPLOYEE
       SUM CURR_SAL AS 'CURRENT'
           FIRST_NAME AS 'FIRST'
       BY EID AS 'SSN'
    -*Concatenate file EMPDATA to EMPLOYEE to form first MATCH answer set.
3.     MORE
       FILE EMPDATA
       RUN
    -*Second MATCH answer set:

4.  FILE TRAINING
       PRINT EXPENSES
5.     BY PIN AS 'SSN'
6.     AFTER MATCH HOLD OLD-OR-NEW
    END

    -*Print merged file:

7.  TABLE FILE HOLD
       PRINT *
    END
```

1. Defines the EMPDATA fields needed for concatenating it to EMPLOYEE.
2. Starts the MATCH and the main request in the concatenation. The main request defines all printing and sorting for the concatenated files. The sort field is called SSN in the resulting file.
3. Concatenates file EMPDATA to EMPLOYEE. This concatenated file becomes the OLD file in the match.
4. Creates the NEW file in the match.
5. Uses an AS phrase to change the name of the sort field in the NEW file to the same name as the sort field in the OLD file.
6. Defines the merge procedure. All records from the NEW file, the OLD file, and both files are included in the final HOLD file.
7. Prints the values from the merged file.

The following report is produced:



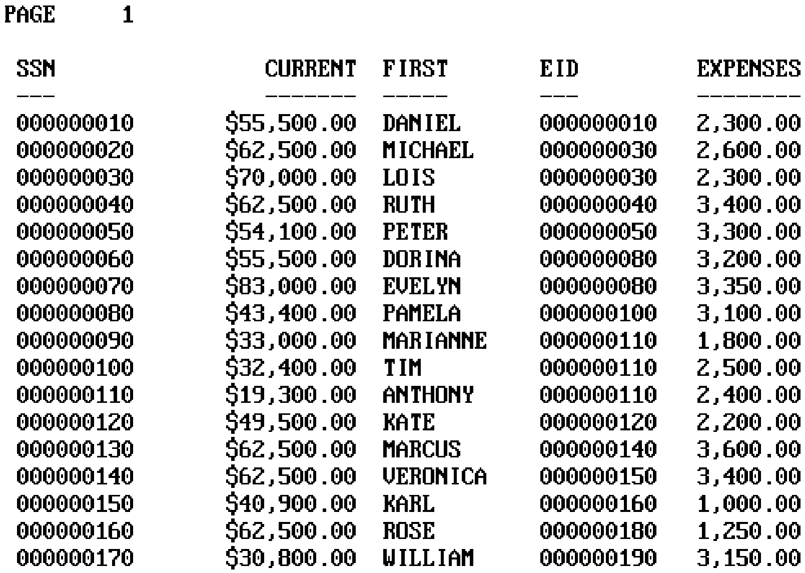
```
PAGE     1


  SSN                  CURRENT  FIRST         EXPENSES
  ---                  -------  -----         --------
  000000010         $55,500.00  DANIEL        2,300.00
  000000020         $62,500.00  MICHAEL              .
  000000030         $70,000.00  LOIS          2,600.00
  000000030         $70,000.00  LOIS          2,300.00
  000000040         $62,500.00  RUTH          3,400.00
  000000050         $54,100.00  PETER         3,300.00
  000000060         $55,500.00  DORINA               .
  000000070         $83,000.00  EVELYN               .
  000000080         $43,400.00  PAMELA        3,200.00
  000000080         $43,400.00  PAMELA        3,350.00
  000000090         $33,000.00  MARIANNE             .
  000000100         $32,400.00  TIM           3,100.00
  000000110         $19,300.00  ANTHONY       1,800.00
  000000110         $19,300.00  ANTHONY       2,500.00
  000000110         $19,300.00  ANTHONY       2,400.00
  000000120         $49,500.00  KATE          2,200.00
  000000130         $62,500.00  MARCUS               .
```

Example Merging Concatenated Data Sources Without a Sort Field

In this example, the merged files do not share a sort field:

```
DEFINE FILE EMPDATA
CURR_SAL/D12.2M = CSAL;
FIRST_NAME/A10 = FN;
EID/A9 = PIN;
END

-*Start MATCH

MATCH FILE EMPLOYEE
SUM CURR_SAL AS 'CURRENT'
    FIRST_NAME AS 'FIRST'
BY EID AS 'SSN'

-*Concatenate EMPDATA to EMPLOYEE to form the first MATCH answer set

MORE
FILE EMPDATA
RUN

-*Second MATCH answer set:

FILE TRAINING
PRINT EXPENSES
BY PIN AS 'EID'
AFTER MATCH HOLD OLD-OR-NEW
END

-*Print merged file:

TABLE FILE HOLD
PRINT *
END
```

The AS phrase changes the answer set. Since the sort fields no longer have the same names, the fields are merged with no regard to matching records.



```
PAGE     1

 SSN                CURRENT  FIRST       EID          EXPENSES
 ---                -------  -----       ---          --------
 000000010       $55,500.00  DANIEL      000000010    2,300.00
 000000020       $62,500.00  MICHAEL     000000030    2,600.00
 000000030       $70,000.00  LOIS        000000030    2,300.00
 000000040       $62,500.00  RUTH        000000040    3,400.00
 000000050       $54,100.00  PETER       000000050    3,300.00
 000000060       $55,500.00  DORINA      000000080    3,200.00
 000000070       $83,000.00  EVELYN      000000080    3,350.00
 000000080       $43,400.00  PAMELA      000000100    3,100.00
 000000090       $33,000.00  MARIANNE    000000110    1,800.00
 000000100       $32,400.00  TIM         000000110    2,500.00
 000000110       $19,300.00  ANTHONY     000000110    2,400.00
 000000120       $49,500.00  KATE        000000120    2,200.00
 000000130       $62,500.00  MARCUS      000000140    3,600.00
 000000140       $62,500.00  VERONICA    000000150    3,400.00
 000000150       $40,900.00  KARL        000000160    1,000.00
 000000160       $62,500.00  ROSE        000000180    1,250.00
 000000170       $30,800.00  WILLIAM     000000190    3,150.00
```

Cartesian Product

Cartesian product enables you to generate a report containing all combinations of non-related records or data instances in the case of a multi-path request. The syntax is

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

where:

`OFF`

Disables Cartesian product. OFF is the default setting.

`ON`

Enables Cartesian product and generates all possible combinations of non-related records.

SET CARTESIAN may also be issued from within a request.

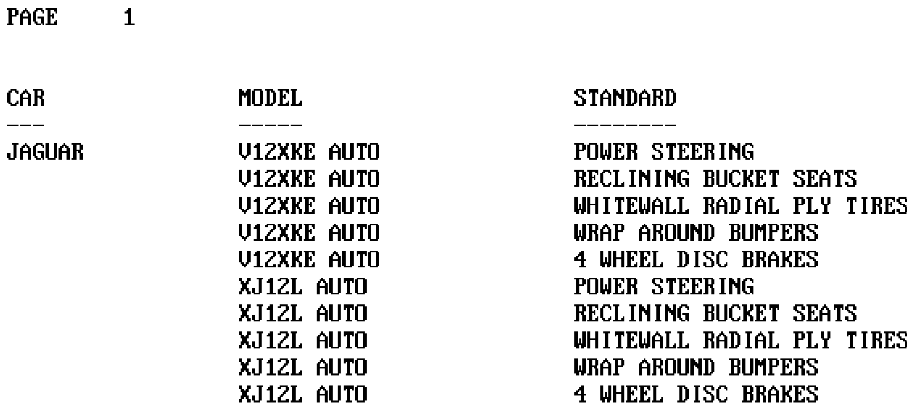
Reference Usage Notes for Cartesian Products

- Cartesian product is performed on the lowest segment common to all paths, whether or not a field in that segment is referenced.
- The SET CARTESIAN command is disabled when ACROSS is specified, and a warning message is issued.
- The SUM display command and the TOT. prefix operator have no effect on Cartesian product.
- SUM, COMPUTE, and WITHIN in combination with the PRINT display command are performed on the Cartesian product.
- ON TABLE COLUMN-TOTAL is automatically generated on the Cartesian product.
- NOSPLIT is disabled if specified in combination with the SET CARTESIAN command, and no warning message is issued.
- MATCH is not supported with the SET CARTESIAN command. A warning message is not issued if MATCH is requested, and the request is processed as if CARTESIAN is set to OFF.
- TABLEF is not supported with the SET CARTESIAN command.

Example Using Cartesian Product

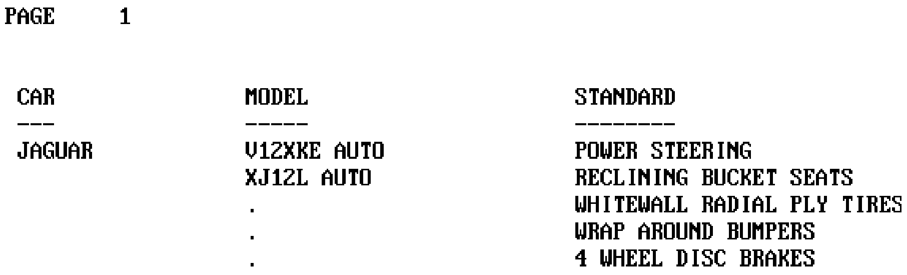
When CARTESIAN is set to ON, the following multi-path request produces a report containing all possible combinations of models and standards for each car:

```
TABLE FILE CAR
PRINT MODEL STANDARD
BY CAR
IF CAR EQ 'JAGUAR'
END
```



```
PAGE     1


CAR               MODEL                     STANDARD
---               -----                     --------
JAGUAR            V12XKE AUTO               POWER STEERING
                  V12XKE AUTO               RECLINING BUCKET SEATS
                  V12XKE AUTO               WHITEWALL RADIAL PLY TIRES
                  V12XKE AUTO               WRAP AROUND BUMPERS
                  V12XKE AUTO               4 WHEEL DISC BRAKES
                  XJ12L AUTO                POWER STEERING
                  XJ12L AUTO                RECLINING BUCKET SEATS
                  XJ12L AUTO                WHITEWALL RADIAL PLY TIRES
                  XJ12L AUTO                WRAP AROUND BUMPERS
                  XJ12L AUTO                4 WHEEL DISC BRAKES
```

When CARTESIAN is set to OFF, the same request results in a report from the CAR file, containing a list of models and standards without logical relationships:



```
PAGE     1


CAR               MODEL                     STANDARD
---               -----                     --------
JAGUAR            V12XKE AUTO               POWER STEERING
                  XJ12L AUTO                RECLINING BUCKET SEATS
                  .                         WHITEWALL RADIAL PLY TIRES
                  .                         WRAP AROUND BUMPERS
                  .                         4 WHEEL DISC BRAKES
```

16 Improving Data Retrieval

Topics:

- Restructuring Data: Alternate Views
- Fast Retrieval: Automatic Indexed Retrieval
- Fast Data Retrieval: TABLEF

In the simplest case, you can report from a single data file that resides on your system and is controlled by your FOCUS session. FOCUS supports the following high-performance methods for accessing data:

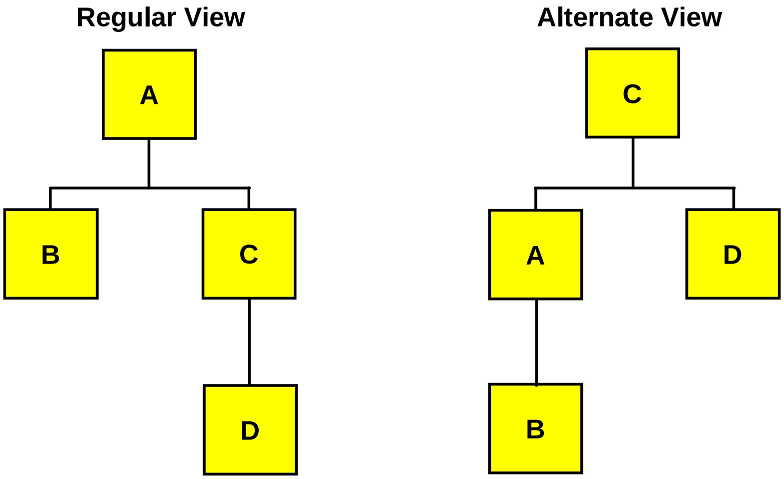
- Taking an alternate view of the data structure. You can temporarily rotate a data file hierarchy and then report from the data file using this alternate view.
- Faster data retrieval. You can use automatic indexed retrieval to take advantage of indexed fields that are used in equality or range tests.
- AUTOPATH. You can use automatic alternate file views with the AUTOPATH feature.
- More efficient data retrieval. You can increase your reporting efficiency by using a pre-sorted data file together with the TABLEF command.

Restructuring Data: Alternate Views

If you are using certain data sources such as IMS, CA-IDMS/DB, or FOCUS, you can rotate the data source, creating an alternate view which changes some of the segment relationships and enables you to access the segments in a different order. By reporting from an alternate view, you can do the following:

- Change the access path. For example, you can access data in a lower segment more quickly by promoting that segment to a higher level.
- Change the path structure of a data file. This option is especially helpful if you wish to create a report using several sort fields that are on different data file paths. By changing the view of the data file hierarchy, all the desired sort fields can be on the same path.

For example, consider the regular and alternate views below:



Since C is the root segment in the alternate view, particular instances of C can be selected faster.

Syntax

How to Request an Alternate View

To request an alternate view, add the name of a field—one found in the alternate root segment—to the file name in the TABLE command, separated by a period (.):

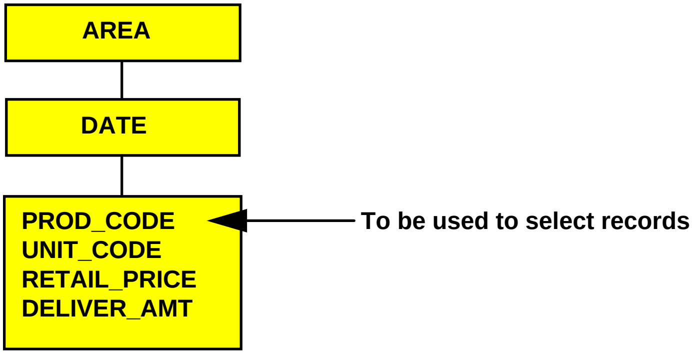
```
TABLE FILE filename.fieldname
```

Reference Usage Notes for Restructuring Data

- If you use a non-indexed field, FOCUS goes through each segment instance until it finds the specified record. This process is, therefore, less efficient than when an indexed field is used.
- When you use the alternate view feature on a particular child segment, the data retrieved from that segment will be retrieved in physical order, not logical order. This is so because the child becomes a root segment for the report request, and data from root segments are retrieved in physical order.
- Alternate view on an indexed field is a special case that causes FOCUS to use the index for retrieval. When you perform an alternate view on an indexed field, you enhance the speed of retrieval. However, you must include an equality test on the indexed field (for example, WHERE MONTH EQ 1 OR 2).
- A field name specified in an alternate file view may not be qualified or exceed 12 characters.
- Automatic Indexed Retrieval (AUTOINDEX) is never invoked in a TABLE request against an alternate file view.

Example Restructuring Data

Consider the following data structure, in which PROD_CODE is an indexed field:



You could issue the following request to promote the segment containing PROD_CODE to a higher level. This promotes the segment containing the field PROD_CODE to the top of the hierarchy.

```
TABLE FILE SALES.PROD_CODE
"SALES OF B10 DISTRIBUTED BY AREA"
SUM UNIT_SOLD AND RETAIL_PRICE
BY AREA
WHERE PROD_CODE EQ 'B10'
ON TABLE COLUMN-TOTAL
END
```

Fast Retrieval: Automatic Indexed Retrieval

FOCUS can automatically take advantage of indexed fields in most TABLE requests that contain equality or range tests on those fields. AUTOINDEX is supported only with FOCUS and Fusion databases.

Syntax How to Use Indexed Retrieval

The syntax is:

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

Valid values are:

`ON`

Uses indexed retrieval when possible. When AUTOINDEX is ON, a request can use indexed retrieval only if it contains an equality or range test against an indexed field that exists in the highest segment referenced in the request. You can use equality and range tests with all data types; however, if you use a range test with packed data, FOCUS performs sequential retrieval.

`OFF`

Sets AUTOINDEX off. When AUTOINDEX is OFF, the only way to accomplish indexed retrieval is with an indexed view. OFF is the default; however, this default may have been changed during installation. You can check your setting by issuing the ? SET command.

Reference Usage Notes for Indexed Retrieval

- AUTOINDEX is never invoked when the TABLE request contains an alternate file view (that is, TABLE FILE *filename.fieldname*).
- Even if AUTOINDEX is ON, FOCUS does not perform indexed retrieval when the TABLE request contains BY HIGHEST or BY LOWEST phrases. These phrases are only performed when the request uses an indexed view.

Example Using Indexed Retrieval

The following Master File is referenced in the examples that follow:

```
FILENAME=SALES,SUFFIX=FOC,
  SEGNAME=STOR_SEG,SEGTYPE=S1,
    FIELDNAME=AREA,ALIAS=LOC,FORMAT=A1,$
  SEGNAME=DATE_SEG,PARENT=STOR_SEG,SEGTYPE=SH1,
    FIELDNAME=DATE,ALIAS=DTE,FORMAT=A4MD, $
  SEGNAME=DEPT,PARENT=DATE_SEG,SEGTYPE=S1,
    FIELDNAME=DEPARTMENT,ALIAS=DEPT,FORMAT=A5,FIELDTYPE=I,$
    FIELDNAME=DEPT_CODE,ALIAS=DCODE,FORMAT=A3,FIELDTYPE=I,$
    FIELDNAME=PROD_TYPE,ALIAS=PTYPE,FORMAT=A10,FIELDTYPE=I,$
  SEGNAME=INVENTORY,PARENT=DEPT,SEGTYPE=S1,$
    FIELDNAME=PROD_CODE,ALIAS=PCODE,FORMAT=A3,FIELDTYPE=I,$
    FIELDNAME=UNIT_SOLD,ALIAS=SOLD,FORMAT=I5,$
    FIELDNAME=RETAIL_PRICE,ALIAS=RP,FORMAT=D5.2M,$
    FIELDNAME=DELIVER_AMT,ALIAS=SHIP,FORMAT=I5,$
```

The following procedure contains an equality test on DEPT_CODE and PROD_CODE. FOCUS uses DEPT_CODE for indexed retrieval since it is in the higher of the referenced segments.

```
SET AUTOINDEX=ON
TABLE FILE SALES
SUM UNIT_SOLD RETAIL_PRICE
IF DEPT_CODE EQ 'H01'
IF PROD_CODE EQ 'B10'
END
```

If your TABLE request contains an equality or range test against more than one indexed field in the same segment, AUTOINDEX uses the first index referenced in that segment for retrieval. The following stored procedure contains an equality test against two indexed fields. Since DEPT_CODE appears before PROD_TYPE in the Master File, AUTOINDEX uses DEPT_CODE for retrieval.

```
SET AUTOINDEX=ON
TABLE FILE SALES
SUM UNIT_SOLD AND RETAIL_PRICE
IF PROD_TYPE EQ 'STEREO'
IF DEPT_CODE EQ 'H01'
END
```

FOCUS does not invoke indexed retrieval if the equality or range test is against an indexed field that does not reside in the highest referenced segment. In the following example, indexed retrieval is not performed because the request contains a reference to AREA, a field in the STOR_SEG segment:

```
SET AUTOINDEX=ON
TABLE FILE SALES
SUM UNIT_SOLD AND RETAIL_PRICE
BY AREA
IF PROD_CODE EQ 'B10'
IF PROD_TYPE EQ 'STEREO'
END
```

When a Request Contains an Indexed View

When a request specifies an indexed view (for example, TABLE FILE *filename.indexed_fieldname*) FOCUS performs indexed retrieval provided an IF or WHERE equality test is present for *indexed_fieldname*:

Fast Data Retrieval: TABLEF

FOCUS provides TABLEF, a variation of the TABLE command that provides a very fast method of retrieving data if the data file is already stored in the order required for printing, and no additional sorting is required.

Using TABLEF, FOCUS retrieves records in the logical sequence from the data file. The standard report request syntax applies, subject to the following rules:

- Any BY phrases must be compatible with the logical sequence of the database. BY phrases are used only to establish control break, not to change the order of the records.
- ACROSS phrases are not permitted.
- Multiple display commands are not permitted. Only one display command may be used.
- After the report is printed, RETYPE, HOLD, and SAVE are not available. However, you can produce an extract file if you include ON TABLE HOLD or ON TABLE SAVE as part of the request.
- NOSPLIT is not compatible with the TABLEF command and produces an FOC037 error message.
- TABLEF can also be used with HOLD or other non-FOCUS data files, when the natural sort sequence of both the request and the data are the same.
- TABLEF is not supported with SET EMPTYREPORT. When a TABLEF request retrieves zero records, EMPTYREPORT behaves as if EMPTYREPORT is set to ON.

Example Printing Using Fast Table Retrieval

For example, if you had previously created a HOLD file of the EMPLOYEE database sorted by the CURR_SAL, LAST_NAME, and FIRST_NAME fields, you could issue the following TABLEF request:

```
TABLEF FILE HOLD
PRINT CURR_SAL AND LAST_NAME AND FIRST_NAME
END
```

17 Creating Extended Matrix Reports

Topics:

- Introduction to Extended Matrix Reports
- Creating Rows From Data: FOR
- Inter-Row Calculations: RECAP
- Creating Recursive Models
- User-Written Routines for Calculations
- Formatting Options
- Supplying Data Directly in the Report Request
- Saving Intermediate Report Results: POST and PICKUP
- Creating HOLD Files From EMR Reports

FOCUS Extended Matrix Reporting (EMR) is designed for the special needs associated with creating, calculating, and presenting financially oriented data such as balance sheets, consolidations, or budgets. These reports distinguish themselves from other FOCUS reports because calculations are inter-row as well as inter-column and each row or line represents a unique entry or series of entries that can be aggregated directly from the input data or calculated as some function of the data.

Introduction to Extended Matrix Reports

EMR is an integrated extension of the FOCUS TABLE command. By adding two statements (FOR and RECAP), you can handle a vastly-expanded range of applications.

Used in conjunction with Dialogue Manager, EMR can be used to perform “what if” scenarios and develop complete decision support systems. These systems can take advantage of FOCUS’ integration and include statistical analyses and graphics, in addition to the standard financial statements.

Procedures using EMR are not hard-wired to the data. Like any other FOCUS request, these procedures can easily be changed. EMR includes the following facilities:

- Row/column formatting: You can easily specify results in a row-by-row, column-by-column fashion. EMR includes a matrix notation so that you can perform calculations between row and column elements or cells (see *Inter-Row Calculations: RECAP* on page 17-12).
- Intermediate results: You can post EMR results to an external file and pick them up at a later time for analysis. This is useful when intermediate results are developed and a final procedure consolidates the results later (see *Saving Intermediate Report Results: POST and PICKUP* on page 17-28).
- Inline data entry: EMR enables you to specify constants from within the procedure in addition to the data values retrieved from your data source (see *Supplying Data Directly in the Report Request* on page 17-28).
- Recursive reports: You can produce reports where the results from the end of one time period or column become the starting balance in the next. For example, you could use recursive reports to produce a cash flow projection (see *Creating Recursive Models* on page 17-21).

Sample Request

The following annotated example illustrates several of the points discussed in *Introduction to Extended Matrix Reports* on page 17-2. Notice the similarity of this example to a typical reporting request, except for the addition of the two new EMR keywords. The example produces a simple asset sheet, contrasting the results of two years. Though the example is relatively simple and the results brief, an EMR model could be quite comprehensive, using Dialogue Manager to create a flexible, interactive model.

```
    TABLE FILE FINANCE
    HEADING CENTER
    "COMPARATIVE ASSET SHEET </2"
    SUM AMOUNT ACROSS HIGHEST YEAR
    WHERE YEAR EQ '1983' OR '1982'
1.  FOR ACCOUNT
2.   1000            AS 'UTILITY PLANT'                  LABEL   UTP       OVER
2.   1010 TO 1050    AS 'LESS ACCUMULATED DEPRECIATION' LABEL   UTPAD     OVER
3.   BAR                                                                   OVER
     RECAP UTPNET=UTP-UTPAD; AS 'TOTAL PLANT-NET'                          OVER
     BAR                                                                   OVER
      2000 TO 3999   AS 'INVESTMENTS'                    LABEL   INV       OVER
      "CURRENT ASSETS"                                                     OVER
      4000           AS 'CASH'                           LABEL   CASH      OVER
     5000 TO 5999    AS 'ACCOUNTS RECEIVABLE-NET'        LABEL   ACR       OVER
      6000           AS 'INTEREST RECEIVABLE'            LABEL   ACI       OVER
      6500           AS 'FUEL INVENTORY'                 LABEL   FUEL      OVER
      6600           AS 'MATERIALS AND SUPPLIES'         LABEL   MAT       OVER
      6900           AS 'OTHER'                          LABEL   MISC      OVER
      BAR                                                                  OVER
     RECAP TOTCAS = CASH+ACR+ACI+FUEL+MAT+MISC ; AS 'TOTAL CURRENT ASSETS' OVER
      BAR                                                                  OVER
      7000           AS 'DEFERRED DEBITS'                LABEL   DEFDB     OVER
      BAR                                                                  OVER
     RECAP TOTAL=UTPNET+INV+TOTCAS+DEFDB; AS 'TOTAL ASSETS'                OVER
      BAR AS '='
      FOOTING CENTER
      "</2 *** PRELIMINARY ASSET SHEET BASED ON UNAUDITED FIGURES ***"
    END
```

1. FOR and OVER are EMR keywords that enable you to easily structure the report on a line-by-line basis.

2. LABEL assigns a variable name to a line item for use in a RECAP calculation.

3. BAR enables you to underline a column of numbers before performing a RECAP calculation.

The report follows:

```
                              COMPARATIVE ASSET SHEET

                                      YEAR
                                                 1983             1982
---------------------------------------------------------------------------
UTILITY PLANT                              $1,430,903       $1,294,611
LESS ACCUMULATED DEPRECIATION                $249,504         $213,225
                                           ----------          -------
TOTAL PLANT-NET                            $1,181,399       $1,081,386
                                           ----------          -------
INVESTMENTS                                      $818           $5,639
CURRENT ASSETS
CASH                                           $4,938           $4,200
ACCOUNTS RECEIVABLE-NET                       $28,052          $23,758
INTEREST RECEIVABLE                           $15,945          $10,206
FUEL INVENTORY                                $35,158          $45,643
MATERIALS AND SUPPLIES                        $16,099          $12,909
OTHER                                          $1,264           $1,743
                                           ----------          -------
TOTAL CURRENT ASSETS                         $101,456          $98,459
                                           ----------          -------
DEFERRED DEBITS                               $30,294          $17,459
                                           ----------          -------
TOTAL ASSETS                               $1,313,967       $1,202,943
                                           ==========       ==========

***PRELIMINARY ASSET SHEET BASED ON UNAUDITED FIGURES ***
```

Sample Databases and Temporary Fields

The examples in this chapter use the LEDGER and REGION databases, which are included on your FOCUS tape or cartridge. Following are several temporary fields that have been defined and used for many of the examples:

```
DEFINE FILE LEDGER
CUR_YR/I5C=AMOUNT;
LAST_YR/I5C=.87*CUR_YR - 142;
NEXT_YR/I5C=1.13*CUR_YR + 222;
MARKS/I5C=AMOUNT;
POUNDS/I5C=3.2*AMOUNT;
END
```

```
DEFINE FILE REGION
CUR_YR=E_ACTUAL;
LAST_YR=.831*CUR_YR;
NEXT_YR=1.2297*CUR_YR;
END
```

```
DEFINE FILE REGION ADD
REGION/A4=IF E_ACTUAL OR E_BUDGET THEN 'EAST' ELSE 'WEST';
END
```

```
DEFINE FILE REGION ADD
AMOUNT/I5C=E_ACTUAL;
END
```

You will need to define these temporary fields in order to reproduce any of the examples included.

Creating Rows From Data: FOR

A normal TABLE request sorts the lines of the report according to the BY phrase you use. The data retrieved is either sorted low-to-high or high-to-low as requested. The lines may be limited by a screening phrase to a specific subset, but:

- They appear in a sort order.
- Lines appear only for values that are retrieved from the file.
- Inserting free text between the rows can only be performed when a sort field changes value, such as:

  ```
  ON DIVISION SUBFOOT
  ```

- Inserting calculations between rows can only be performed when a sort field changes value, such as:

  ```
  ON DIVISION RECAP
  ```

By contrast, the FOR statement allows you to structure your report line by line.

Example Creating Rows From Data

Assume you have a simple FOCUS database with financial data for each corporate account, as follows:

```
CHART OF ACCOUNTS

ACCOUNT           DESCRIPTION

1010              CASH ON HAND
1020              DEMAND DEPOSITS
1030              TIME DEPOSITS
1100              ACCOUNTS RECEIVABLE
1200              INVENTORY
.                      .
.                      .
.                      .
```

Using the FOR phrase in EMR, you can issue the following TABLE request

```
TABLE FILE LEDGER
SUM AMOUNT
FOR ACCOUNT
1010 OVER
1020 OVER
1030 OVER
1100 OVER
1200
END
```

to produce the following report:

```
           AMOUNT
          -------
1010        8,784
1020        4,494
1030        7,961
1100       18,829
1200       27,307
```

Syntax How to Specify Rows

The syntax for specifying a fixed set of rows is

```
FOR fieldname [NOPRINT]
value [OR value OR...] OVER
.
.
[value OR value]
END
```

where:

field

Is a field name in the data source.

value

Is the value on which you are reporting.

Note: A tag value (like 1010) may be referred to only once in an EMR statement. You can, however, use the values retrieved more than once (see *Row Labels* on page 17-13).

Changing Row Titles: The AS Phrase

Using the AS phrase, you can assign a label for each row of the report that is different from the tag. For example:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND' OVER
1020 AS 'DEMAND DEPOSITS'
END
```

Note: The single quotation marks are necessary only if the phrase you are using has embedded blanks.

This produces the following report:

```
                     AMOUNT
                     ------
CASH ON HAND          8,784
DEMAND DEPOSITS       4,494
```

If no AS phrase is included, FOCUS will display the first value as a label in the report.

Creating Rows From Multiple Records: OR, TO, and Masks

There are different ways to combine multiple records from your FOCUS database into an EMR report line:

- Use the OR phrase to sum the values of two or more tags in a single expression.
- Use the TO phrase to identify a range of values on which to report.
- Use a mask to pick up a group of tag values without having to specify each one.

  FOCUS looks first for an exact reference, next for a range, and finally for a mask.

Summing Values in Rows: The OR Phrase

Use the OR phrase within the FOR command to sum the values of two or more tags in a single report line. The syntax is:

```
FOR fieldname
value OR value [OR value...]
.
.
.
```

For example, the following model

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 OR 1020 OR 1030  AS 'CASH'                  OVER
1100                  AS 'ACCOUNTS RECEIVABLE'    OVER
1200                  AS 'INVENTORY'
END
```

produces this report:

```
                            AMOUNT
                            ------
CASH                        21,239
ACCOUNTS RECEIVABLE         18,829
INVENTORY                   27,307
```

Identifying a Range of Values: The TO Phrase

Instead of using a specific tag for a report line, you can identify a range of tag values with the TO phrase in the FOR statement. The syntax is

```
tagvalue1 TO tagvalue2
```

where:

`tagvalue1`
: Is the lower limit of the range.

`TO`
: Is the required keyword.

`tagvalue2`
: Is the upper limit of the range.

For example, since CASH accounts in the LEDGER system are accounts 1010, 1020, 1030, you can write:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 TO 1030 AS 'CASH'
END
```

Masking Tags

If the tag field is alphanumeric, you can perform a match with the database value by using a masked match. Use the dollar sign character ($) as the mask. For instance, the tag

```
A$$D
```

matches any four-character value beginning with A and ending with D. The two middle places can be anything. This is useful for picking up a whole group of tag values without having to specify each one. For example:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
10$$ AS 'CASH'                 OVER
1100 AS 'ACCOUNTS RECEIVABLE'  OVER
1200 AS 'INVENTORY'
END
```

In the example, FOCUS will look for any four-character accounts that begin with 10, and will use those amounts to produce the CASH row of the report.

Using Tags From External Files

The tags for a row of an EMR report can come from an external file. Consider the file CASHSTUF, that contains the following tags:

```
1010
1020
1030
```

The following TABLE request uses the tags from the external file:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
(CASHSTUF) AS 'CASH'                 OVER
1100                  AS 'ACCOUNTS RECEIVABLE'
END
```

Notice that the file name must be enclosed in parentheses.

The amounts in accounts 1010, 1020, and 1030 are summed into the CASH line of the EMR report.

Using the BY Phrase

The word FOR can only appear once in a TABLE request. It substitutes in part for a BY phrase, which controls the sort sequence. However, there can still be up to 32 BY phrases in the request. In general, BY phrases are the major (outer) sort fields in EMR, and FOR is the minor (inner) sort field.

This is illustrated in the following example

```
TABLE FILE REGION
HEADING CENTER
"CURRENT ASSETS FOR REGION <REGION"
" "
SUM CUR_YR LAST_YR
BY REGION NOPRINT AND PAGE-BREAK
FOR ACCOUNT
10$$ AS 'CASH'                      OVER
1100 AS 'ACCOUNTS RECEIVABLE'       OVER
1200 AS 'INVENTORY'                 OVER
BAR                                 OVER
RECAP CUR_ASSET/I5C = R1 + R2 + R3;
END
```

which produces this report:

```
CURRENT ASSETS FOR REGION EAST

                           CUR_YR         LAST_YR
                           ------         -------

CASH                     9,511.00        7,903.64
ACCOUNTS RECEIVABLE          .               .
INVENTORY                    .               .
                          -------          ------
CUR_ASSET                   9,511           7,903
```

Note that a new report page is produced for each region in the company.

The value of a sort control field can be used in a RECAP statement to allow the model to take different actions within each major sort break. For instance, in the above example, a calculation of

```
RECAP X=IF REGION EQ 'EAST' THEN .25*CASH ELSE 0; AS 'AVAILABLE FOR DIVIDENDS'
```

would compute a non-zero value only for the EAST region.

Inter-Row Calculations: RECAP

The RECAP statement allows you to perform calculations on data in the rows of the report to produce new rows. The syntax is

```
RECAP name[/format]=expression;
[AS 'text']
```

where:

`RECAP`
Is the command name and is required. It should begin on a line by itself.

`name`
Is the name you assign to the calculation. The name can be up to 66 characters long, and must start with an alphabetic character.

`format`
Is the field's USAGE format. It cannot exceed the column width. The default is the format of the column in which the computed value will be displayed.

`expression`
Can be any calculation available with the DEFINE command (including IF … THEN … ELSE syntax, special functions, and user-written functions; excluding DECODE, EDIT, and fields in date format). The expression may extend to as many lines as it requires; a semicolon is required at the end of the expression.

`AS '`*`text`*`'`
Allows you to assign a different name to the RECAP expression for the report. Enclose the text in single quotation marks.

Note:

- RECAP expressions refer to other lines in the model using their labels (either explicit or default). Labels referred to in a RECAP expression must also be specified in the report request. Labels are described in *Row Labels* on page 17-13 with examples of appropriate RECAP commands.

  For example:

  ```
  RECAP TOTVAL/D6.2S=IF R1 GT R4 THEN R4 ELSE R1;
  AS 'REDUCED VALUE'
  ```

  In the example, TOTVAL/D6.2S displays the result as six digits with two decimal places (and prints blanks if the value was zero) in each column of the report, regardless of the format of the data in the column. This is useful for printing percentages, for example, in a column of whole numbers.

- Subtotals are not supported in EMR.

Row Labels

Since RECAP calculations are performed among rows, each row in the calculation must be uniquely identified. You may:

- Use the default (implicit) label assigned by EMR.
- Create a row label of your own.
- Mix implicit and explicit labels in a calculation expression.

Note: The RETYPE command with field format redefinition does not recognize labels in EMR.

Using Default (Implicit) Row Labels

EMR assigns a default label to each line that begins with a tag or a RECAP command. These default labels are automatically prefixed with the letter R and are positional, so that the first such line in the model is R1, the second is R2, etc.

You can use these labels to refer to lines in RECAP expressions, as shown in the following example:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND'       OVER
1020 AS 'DEMAND DEPOSITS'    OVER
1030 AS 'TIME DEPOSITS'      OVER
BAR                          OVER
RECAP TOTCASH = R1 + R2 + R3; AS 'TOTAL CASH'
END
```

Account 1010 is assigned the implicit label R1; account 1020, the implicit label R2; and Account 1030, the implicit label R3. The RECAP line is assigned the implicit label R4. Note that *no* label is assigned to the BAR line (or to any line of free text). The report appears as follows:

```
                                AMOUNT
                                ------
CASH ON HAND                     8,784
DEMAND DEPOSITS                  4,494
TIME DEPOSITS                    7,961
                                ------
TOTAL CASH                      21,239
```

Assigning Your Own (Explicit) Row Labels

You can assign a label of up to 66 characters to every tagged and RECAP line. The syntax is:

```
tag AS 'text' LABEL label OVER
```

Labels on tag or data lines must be unique. Labels cannot have blanks or special characters. For example:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
10$$ AS 'CASH'                 LABEL CASH  OVER
1100 AS 'ACCOUNTS RECEIVABLE'  LABEL AR    OVER
1200 AS 'INVENTORY'            LABEL INV   OVER
BAR                                        OVER
RECAP CURASST/I5C= CASH + AR + INV;
END
```

The report follows:

```
                              AMOUNT
                              ------
CASH                          21,239
ACCOUNTS RECEIVABLE           18,829
INVENTORY                     27,307
                              ------
CURASST                       67,375
```

A label is not needed on a RECAP calculation line because the name on the left of the equal sign is used as a label. For example, the following RECAP statement could be referred to by the name CURASST:

```
RECAP CURASST/I5=CASH+AR+INV;
```

Using Labels to Repeat Rows

In certain cases, you may wish to repeat an entire row later in your report. For example, the CASH account can appear in the Asset statement and Cash Flow statements of a financial analysis, as shown below:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
"ASSETS"                         OVER
10$$ AS 'CASH' LABEL TOTCASH     OVER
.
.
"CASH FLOW"                      OVER
RECAP SAMECASH/I5C=TOTCASH; AS 'CASH'
END
```

When you refer to the CASH row the second time, use a RECAP (with a new name) and refer to the label, either explicitly (such as TOTCASH), or implicitly (such as R1) in the row where CASH was first used.

Column Declarations

An EMR report can refer to explicit columns as well as explicit rows. Consider the following request:

```
TABLE FILE LEDGER
SUM     CUR_YR AS 'CURRENT,YEAR'
    LAST_YR AS 'LAST,YEAR'
COMPUTE CHANGE/I5C = CUR_YR - LAST_YR;
FOR ACCOUNT
1010 AS 'CASH ON HAND'                  OVER
1020 AS 'DEMAND DEPOSITS'               OVER
1030 AS 'TIME DEPOSITS'                 OVER
BAR                                     OVER
RECAP TOTCASH/I5C = R1 + R2 + R3; AS 'TOTAL CASH'
END
```

Notice that the EMR matrix contains four rows and three columns of data. Both columns of the report as well as the individual cells of the matrix can be referenced in building an EMR report.

```
                               CURRENT           LAST
                               YEAR              YEAR         CHANGE
                               -------           ----         ------
CASH ON HAND                     8,784          7,214          1,570
DEMAND DEPOSITS                  4,494          3,482          1,012
TIME DEPOSITS                    7,961          6,499          1,462
                                 -----          -----          -----
TOTAL CASH                      21,239         17,195          4,044
```

From the report above, for example, you could use the value 6,499 by referring to column 2, row 3. This is described in *Using Matrix Notation* on page 17-19.

Referring to Column Numbers

A calculation may be performed for only one, or a specific set of columns. To identify the columns, place the column number in parentheses after the label name. For example:

```
TABLE FILE LEDGER
SUM CUR_YR AS 'CURRENT,YEAR'
LAST_YR AS 'LAST,YEAR'
FOR ACCOUNT
1010 AS 'CASH ON HAND'                             OVER
1020 AS 'DEMAND DEPOSITS'                          OVER
1030 AS 'TIME DEPOSITS'                            OVER
BAR                                                OVER
RECAP TOTCASH/I5C = R1 + R2 + R3; AS 'TOTAL CASH'  OVER
" "                                                OVER
RECAP GROCASH(2)/F6.2=100*TOTCASH(1)/TOTCASH(2) - 100;
AS 'CASH GROWTH(%)'
END
```

Note:

- TOTCASH(1) refers to total cash in column 1.
- TOTCASH(2) refers to total cash in column 2.
- The resulting calculation is printed in column 2 of the row labeled CASH GROWTH(%).

The output follows:

```
                        CURRENT         LAST
                        YEAR            YEAR
                        -------         ----
CASH ON HAND              8,784        7,214
DEMAND DEPOSITS           4,494        3,482
TIME DEPOSITS             7,961        6,499
                        -------        -----
TOTAL CASH               21,239       17,195

CASH GROWTH(%)                         23.52
```

After data retrieval is completed, FOCUS calculates a single column all at once, and multiple columns one by one.

Recapping Over Contiguous Columns

When a set of contiguous columns is needed within a RECAP, separate the first and last column numbers with commas. For example, DIFFERENCE (2,5) means to compute the results for columns 2 through 5.

In the following example, the RECAP occurs only for columns 2 and 3.

```
TABLE FILE LEDGER
SUM NEXT_YR CUR_YR LAST_YR
FOR ACCOUNT
10$$ AS 'CASH'                    OVER
1100 AS 'ACCOUNTS RECEIVABLE'     OVER
1200 AS 'INVENTORY'               OVER
BAR                               OVER
RECAP ATOT(2,3)/I5C = R1 + R2 + R3;
AS 'ASSETS--ACTUAL'
END
```

```
                            NEXT_YR      CUR_YR       LAST_YR
                            -------      ------       -------
CASH                         25,991      21,239        17,195
ACCOUNTS RECEIVABLE          21,941      18,829        15,954
INVENTORY                    31,522      27,307        23,329
                             ------      ------        ------
ASSETS--ACTUAL                           67,375        56,478
```

Column Addressing

When you need a calculation not for every column, but for every other, or every third column, you can supply a factor to do this. The left-hand side of the expression has the form

`name(s,e,i)/format=`

where:

s

Is the starting column.

e

Is the ending column (may be * to denote all columns).

i

Is the increment factor.

Column addressing is useful when several data fields are displayed within each value of a column sort. For example, in the statement

```
SUM ACTUAL AND FORECAST ACROSS MONTH
```

there are two columns for each month. If a calculation is to be performed only for the ACTUAL data, you can control the placement of the results with a RECAP of the form:

```
RECAP VALUE(1,*,2)=expression;
```

The asterisk means to continue the RECAP for *all* odd-numbered columns (beginning in column 1, with an increment of 2, for all columns).

Relative Column Addressing

A calculation can refer to a column plus or minus 1, 2, etc., relative to a starting point. Use an asterisk (*) to indicate “this column”—that is, the column to which the reference is being made. Note that the reference to COMP=FIX(*); is equivalent to COMP=FIX;. For example

```
COMP=FIX(*)-FIX(*-1);
```

may refer to the change in fixed assets from one period to the next.

In the following example, the change in cash (CHGCASH) is computed for columns 1 and 2. The last RECAP places a blank in column 3. Otherwise, that value would be computed as equivalent to cash for the variable LAST_YR (since TOTCASH(4) is evaluated as 0).

```
TABLE FILE LEDGER
SUM NEXT_YR CUR_YR LAST_YR
FOR ACCOUNT
10$$ AS 'TOTAL CASH' LABEL TOTCASH           OVER
" "                                          OVER
RECAP CHGCASH/I5SC = TOTCASH(*) - TOTCASH(*+1);
AS 'CHANGE IN CASH'                          OVER
RECAP CHGCASH(3)=0;
END
```

```
                       NEXT_YR        CUR_YR       LAST_YR
                       -------        ------       -------
TOTAL CASH              25,991        21,239        17,195

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

Referring to a Column by Its Value

When the ACROSS option is used in a report, all of the retrieved values are aligned under their appropriate columns. Each column has a name that is displayed above it. For example

```
SUM AMOUNT ACROSS YEAR
.
.
.
```

produces a report with the across title YEAR. The entire column of data that has a specific value (YEAR, in this case) can be directly addressed in a RECAP. For instance:

```
RECAP FACTOR=IF YEAR LT 1975 THEN .09*COST ELSE ESTCOST;
```

Adding Additional Columns

The number of columns in any report is controlled by the request statement. For instance, if the verb phrase is SUM AMOUNT AND FORECAST, there are two columns: AMOUNT and FORECAST. Additional columns can be created in EMR just as on regular FOCUS reports using the COMPUTE statement.

A computed column may specify a calculation, or just allocate the space, column title, and format. The following example performs the designated calculation on each tagged or RECAP line of the report. The RECAP lines, however, may change the calculation.

```
SUM CUR_YR AND LAST_YR
COMPUTE DIFFERENCE/D8S=CUR_YR - LAST_YR ;
FOR ACCOUNT
.
.
.
END
```

To add an allocated column without a calculation, use the following syntax:

```
COMPUTE fieldname/format= ;
```

For example, to add one or more future time periods to a report:

```
SUM AMOUNT ACROSS YEAR AND COMPUTE 1979=;
WHERE YEAR GE '1972' AND YEAR LE '1978'
.
.
.
```

Using Matrix Notation

A row and column can be addressed in an expression by the notation

```
E(r,c)
```

where:

`E`

Is a required constant.

`r`

Is the row number.

`c`

Is the column number. Use an asterisk (*) to indicate the current column.

Consider the following example:

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

The report follows:

```
                 E_ACTUAL      E_BUDGET       W_ACTUAL      W_BUDGET
                 --------      --------       --------      --------
SALES               6,000         4,934          7,222         7,056
COST                4,650         3,760          5,697         5,410
                    -----         -----          -----          ----
PROFIT              1,350         1,174          1,525         1,646

EAST--VARIANCE     176
WEST--VARIANCE                             -121
```

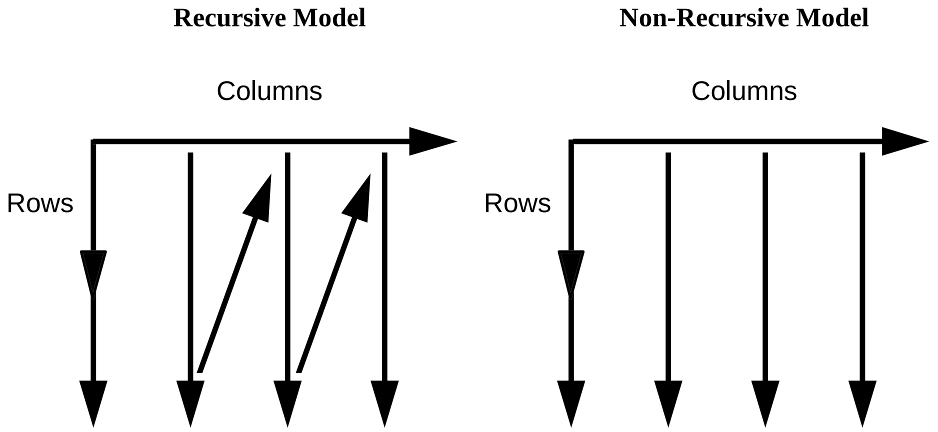
Note: Both labeled and default notation can be used in the same expression.

Creating Recursive Models

Models involving different time periods often require using the ending value of one time period as the starting value for the next time period. The series of calculations describing these situations has two characteristics:

- The labels on one or more RECAP lines are duplicates of other lines. That is, they are used repeatedly to recompute some values.
- A calculation may refer to a label not yet described, but provided later in the model. If, at the end of the model, a label that is needed is missing, an error message is displayed.

Recursive models require that the columns are produced in sequential order, one by one. In nonrecursive models, all of the columns can be produced simultaneously. Schematically, these patterns are shown below.



EMR automatically switches to sequential order as soon as either of the two modeling conditions requiring the switch is recognized (that is, either reuse of labels by different lines, or forward reference to a label in a calculation).

Example Recursive Models

The following example illustrates recursive models. Note that one year’s ENDCASH becomes the next year’s STARTING CASH.

```
TABLE FILE REGION
SUM LAST_YR CUR_YR NEXT_YR
FOR ACCOUNT
10$$ AS 'STARTING CASH' LABEL STCASH        OVER
RECAP STCASH(2,*) = ENDCASH(*-1);           OVER
" "                                         OVER
3000 AS 'SALES' LABEL SLS                   OVER
3100 AS 'COST' LABEL COST                   OVER
BAR                                         OVER
RECAP PROFIT/I5C = SLS - COST;              OVER
" "                                         OVER
RECAP ENDCASH/I5C = STCASH + PROFIT;
END
```

The report appears as:

```
PAGE 1

                       LAST_YR          CUR_YR        NEXT_YR
                       -------          ------        -------

STARTING CASH            7,903           9,024         10,374

SALES                    4,985           6,000          7,378
COST                     3,864           4,650          5,718
                       -------          ------        -------
PROFIT                   1,121           1,350          1,660

ENDCASH                  9,024          10,374         12,034
```

Note: Under CMS, the above example generates a warning message.

User-Written Routines for Calculations

You may provide your own calculation routines in RECAP lines to perform special-purpose calculations, a useful feature when these calculations are mathematically complex or require extensive look-up tables. User-written routines are coded as subroutines in any language that supports a call process, such as FORTRAN, COBOL, PL/1, and BAL. The syntax for calling the routine is

```
routine(arg1, arg2, ... , argn, 'format')
```

where:

routine

Is the eight-character name of the routine. It must be different from any row label and cannot contain any of the following special characters:
= -, / ( ).

arg1...argn

Are the input arguments. They are row labels, and they can be indexed with column numbers to refer to specific columns. Make sure that the values being passed to the subroutine agree in number and type with the arguments as coded in the subroutine.

format

Is the format of the return value, which must be enclosed in single quotation marks.

Chapter 9, *Using Functions and Subroutines*, has complete information about writing your own routines.

Formatting Options

Using EMR, you can add rows to your report which do not specifically access your FOCUS database. These new rows can be any one of the following:

- Free text (see *Inserting Free Text* on page 17-24).
- Underlines (see *Underlining: BAR* on page 17-25).
- Calculations based upon data previously retrieved (see *Inter-Row Calculations: RECAP* on page 17-12).
- New column titles with the AS phrase (see *Changing Row Titles: The AS Phrase* on page 17-8).

```
TABLE FILE LEDGER
SUM ACCOUNT FOR ACCOUNT
“ --CASH ACCOUNTS-- “ OVER ◄──────────────────────────── Free Text
10$$ AS ‘CASH’ OVER◄──────────────────────────── Assigning Text
.
.
OVER BAR◄─────────────────────────────────── Underlining
END
```

Other EMR formatting options include:

- Suppressing the printing of tagged lines. See *Suppressing Tagged Lines* on page 17-26.
- Suppressing the printing of lines with no data. See *Suppressing Lines With No Data* on page 17-27.
- Specifying page breaks. See *Specifying a Page Break* on page 17-27.

Inserting Free Text

You can insert text anywhere in your report by typing it on a line by itself and enclosing it within double quotation marks. In the following example, three lines of free text are inserted, one blank and two text lines:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
" --- CASH ACCOUNTS ---"            OVER
1010 AS 'CASH ON HAND'              OVER
1020 AS 'DEMAND DEPOSITS'           OVER
1030 AS 'TIME DEPOSITS'             OVER
" "                                 OVER
" --- OTHER CURRENT ASSETS ---"     OVER
1100 AS 'ACCOUNTS RECEIVABLE'       OVER
1200 AS 'INVENTORY'
END
```

The report prints as follows:

```
                                 AMOUNT
                                 ------
--- CASH ACCOUNTS ---
CASH ON HAND                      8,784
DEMAND DEPOSITS                   4,494
TIME DEPOSITS                     7,961

--- OTHER CURRENT ASSETS ---
ACCOUNTS RECEIVABLE              18,829
INVENTORY                        27,307
```

Notice that the blank line was created by enclosing a blank within double quotation marks on a separate line of the report request.

Free lines of text can also contain data developed in your EMR report (see *Inserting Data Variables Into Text Lines* on page 17-26).

Underlining: BAR

Reports with columns of numbers frequently need to use an underline before some recap calculations. You can specify the underline character of your choice by using the word BAR in place of the tag value. The syntax is

```
BAR [AS 'character'] OVER
```

where:

character

Is any character you choose. Enclose it in single quotation marks. The default character is the hyphen (-).

For example:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND'        OVER
1020 AS 'DEMAND DEPOSITS'     OVER
1030 AS 'TIME DEPOSITS'       OVER
BAR                           OVER
RECAP TOTCASH = R1 + R2 + R3;
END
```

The report output appears as:

```
                           AMOUNT
                           ------
CASH ON HAND                8,784
DEMAND DEPOSITS             4,494
TIME DEPOSITS               7,961
                           ------
TOTCASH                    21,239
```

The BAR … OVER statement underlines only the column containing the verb object.

The RECAP statement is described in *Inter-Row Calculations: RECAP* on page 17-12.

Inserting Data Variables Into Text Lines

To include a data variable in a text line, include its label (either default or explicit) in the text line. The syntax is

```
"text <label[(c)][>]"
```

where:

`<`
: Is a required left caret to bracket the label.

`label`
: Is the explicit or implicit row label.

`c`
: Is an optional cell identifier that indicates the column number of the cell. This identifier, however, is required whenever there is more than one column in the report. If you use it, enclose it in parentheses.

`>`
: Is an optional right bracket (see Chapter 10, *Customizing Tabular Reports*, for a description of the effects of the right bracket).

The following text line will retrieve the value contained in column 1 for the row labeled TOTCASH:

```
"TOTAL CASH IN THE CURRENT YEAR IS <TOTCASH(1)"
```

Suppressing Tagged Lines

Sometimes you may retrieve tagged lines solely for use in a calculation, that is, you do not want them to print on the report. To suppress printing these, add the word NOPRINT to the line declaration, as you would in a TABLE request. For example:

```
TABLE FILE REGION
SUM AMOUNT FOR ACCOUNT
3000 AS 'SALES' LABEL SLS             OVER
3100 AS 'COST' LABEL COST NOPRINT     OVER
RECAP PROFIT/I5C = SLS - COST;        OVER
" "                                   OVER
RECAP ROS/F6.2=100*PROFIT/SLS;
AS 'RETURN ON SALES'
END
```

The report follows:

```
                             AMOUNT
                             ------
SALES                         6,000
PROFIT                        1,350

RETURN ON SALES               22.50
```

You can also suppress printing of intermediate RECAP lines used as input to other calculations by adding the word NOPRINT to the RECAP statement after the semicolon.

Suppressing Lines With No Data

The text for a tagged line is printed even if no data is found in the file for the tag values, with a period (.) representing the missing data. You can override this convention by adding the phrase WHEN EXISTS to the definition of a tagged line. This makes printing a line dependent upon the existence of data for the tag. This feature is particularly useful, for example, when the same model is applied to different divisions in a company.

In the following example, assume Division 1 is a real estate syndicate and Division 2 is a bank; their balance sheets can be described in one EMR report. Lines that are irrelevant for each division will not be printed.

```
TABLE FILE LEDGER
HEADING CENTER
"BALANCE SHEET FOR DIVISION <DIVISION"
" "
SUM AMOUNT
BY DIVISION NOPRINT AND PAGE-BREAK
FOR ACCOUNT
2000 AS 'LAND' WHEN EXISTS LABEL LD          OVER
2100 AS 'CAR LOANS' WHEN EXISTS LABEL LOAN   OVER
    .
    .
    .
```

Specifying a Page Break

You can request a new page at any point in a report by placing the word PAGE-BREAK in place of the tag value. The syntax is:

```
PAGE-BREAK OVER
```

Supplying Data Directly in the Report Request

In certain cases, you may need to include some additional constants (for example, exchange rates, inflation rates, etc.) in your model. Not all data values for the model have to be retrieved from the file. Using EMR, you can supply data directly in the request statement. The syntax is

```
DATA value,[..., value],$
[AS 'text'] [LABEL label]
```

where:

value

Specifies the value(s) that you are supplying. Values in a list must be separated by commas. The list must end with a comma and a dollar sign (,$).

In the following example, two values (.53 and 1.37) are provided for the exchange rates of marks and pounds, respectively:

```
TABLE FILE LEDGER
SUM MARKS AS 'GERMAN,DIVISION'
POUNDS AS 'ENGLISH,DIVISION'
FOR ACCOUNT
1010 AS 'CASH--LOCAL CURRENCY' LABEL CASH              OVER
DATA .53 , 1.37 ,$ AS 'EXCHANGE RATE' LABEL EXCH   OVER
RECAP US_DOLLARS/I5C= CASH * EXCH;
END
```

The values supplied are taken one column at a time for as many columns as the report originally specified.

Saving Intermediate Report Results: POST and PICKUP

Many reports require results developed in prior reports. This can only be accomplished if some place is provided for storing intermediate values. An example is the need to compute net profit in an Income Statement prior to calculating equity in a Balance Sheet. EMR can save selected lines from one or more models by posting them to a work file. These lines can then be recaptured by picking them up from the work file.

It is not possible to prepare a report entirely from immediate data, but all of the data could be stored in a comma-delimited file.

Posting Data

The syntax for saving any tag, RECAP, or data line, is to add the phrase

```
POST [TO ddname]
```

to the line. The default work file is FOCPOST. Note that this is the common FOCUS comma-delimited file format so that you can, in fact, report from it directly if a FOCPOST Master File is available.

The line will be processed in the usual manner, depending on its other options, and then posted. The label of the line is written first, followed by the numerical values of the columns, each comma-separated, and ending with the terminator character of a dollar sign ($). Note also that a CMS FILEDEF, TSO ALLOCATE, or DYNAM ALLOCATE command must be issued before the request is activated.

```
TABLE FILE LEDGER
SUM CUR_YR LAST_YR
FOR ACCOUNT
1100 LABEL AR POST TO LEDGEOUT OVER
1200 LABEL INV POST TO LEDGEOUT
END
```

At the end of the report, there are two lines in the LEDGEOUT file:

```
AR            , 18829,          15954,$
INV           , 27307,          23329,$
```

Picking Up the Data

You can retrieve posted lines from any work file and use them as if they were provided in a DATA line by adding the phrase:

```
DATA PICKUP [FROM ddname] id1 [OR id2....] [LABEL label] [AS 'text']
```

You can include LABEL and AS phrases, but WHEN EXISTS is not supported.

In the following example, the data in the LEDGER database and the LEDGEOUT file just posted will be used in the RECAP. Note also that you must issue a CMS FILEDEF command, TSO ALLOCATE, or DYNAM ALLOCATE command before the request is activated:

```
TABLE FILE LEDGER
SUM CUR_YR LAST_YR
FOR ACCOUNT
1010 TO 1030 AS 'CASH' LABEL CASH       OVER
DATA PICKUP FROM LEDGEOUT AR
AS 'ACCOUNTS RECEIVABLE' LABEL AR       OVER
DATA PICKUP FROM LEDGEOUT INV
AS 'INVENTORY' LABEL INV                OVER
RECAP CUR_ASSET/I5C = CASH + AR + INV;
END
```

```
                          CUR_YR       LAST_YR
                          ------       -------
CASH                      21,239        17,195
ACCOUNTS RECEIVABLE       18,829        15,954
INVENTORY                 27,307        23,329
                          ------       -------
CUR_ASSET                 67,375        56,478
```

The following line could be used to pick up the sum of the two accounts from LEDGEOUT DATA:

```
DATA PICKUP FROM LEDGEOUT AR OR INV
AS 'ACCTS REC AND INVENTORY'
```

Note: Since the lines in a PICKUP file are in standard comma-delimited format, they could have been provided either from a prior posting, or directly by a user.

Creating HOLD Files From EMR Reports

A report created with EMR can be held as a HOLD file in the same way as all other FOCUS reports (see Chapter 12, *Saving and Reusing Your Report Output*). In this case, you identify the set of tag values specified for each line by the description field (the AS text supplied in the model). When no text is given for a line, the first tag value is used automatically. Therefore, in simple models with only one tag per line and no text, the lines in the HOLD file contain the single tag value. The RECAP calculation lines are computed and form part of the HOLD file. Pure text lines (including BAR lines) are omitted.

The advantage of this feature is the creation of new lines in the HOLD file which are the result of calculations. The augmented HOLD file may then be useful in a variety of TABLE requests.

Note: RECAP lines cannot be reformatted when creating HOLD files.

Example Creating a Hold File

In the following example, the request creates a HOLD file which contains records for CASH, ACCOUNTS RECEIVABLE, INVENTORY, and the RECAP line CURRENT ASSETS:

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 TO 1030 AS 'CASH'                                OVER
1100 AS 'ACCOUNTS RECEIVABLE'                         OVER
1200 AS 'INVENTORY'                                   OVER
RECAP CA = R1 + R2 + R3; AS 'CURRENT ASSETS'
ON TABLE HOLD
END
```

You can query the HOLD file:

```
>
? hold

DEFINITION OF CURRENT HOLD FILE

     FIELDNAME          ALIAS          FORMAT

                        E01            A19
     AMOUNT             E02            I5C
```

You can then report from the HOLD file as

```
TABLE FILE HOLD
PRINT E01 E02
END
```

and produce the following report:

```
                                AMOUNT
                               -------
CASH                            21,239
ACCOUNTS RECEIVABLE             18,829
INVENTORY                       27,307
CURRENT ASSETS                  67,375
```

18 Creating Free-Form Reports

Topics:

- Introduction to Free-Form Reports
- Designing the Report
- Executing a Free-Form Report From a Stored Procedure

This chapter shows you how to present your FOCUS data in an unrestricted or free-form format using a layout of your own design.

Free-form reporting is beneficial when your goal is to present a detailed picture of a database record on each page of the report. Where columnar and matrix reporting present data in columns and rows for purposes of comparison across records, and graphic reporting presents data visually using charts and graphs, the purpose of free-form reporting is to offer you full control over positioning the data on a page.

Introduction to Free-Form Reports

You can design a free-form report by creating a TABLE request that omits the display commands that control columnar and matrix formatting (PRINT, LIST, SUM, and COUNT). Instead, the request includes the following reporting features:

Header feature Contains the body of the report. It displays the text characters, graphic characters, and data fields that make up the report.

Footer feature Contains the footing of the report. This is the text that appears at the bottom of each page of the report. Footers may display the same characters and fields as headers.

Prefix operators Used to indicate field calculations and manipulation.

Temporary fields Used to derive new values from existing data fields.

BY phrases Used to specify the report’s sort order and determine how many records are included on each page.

WHERE phrases Used to select records for the report.

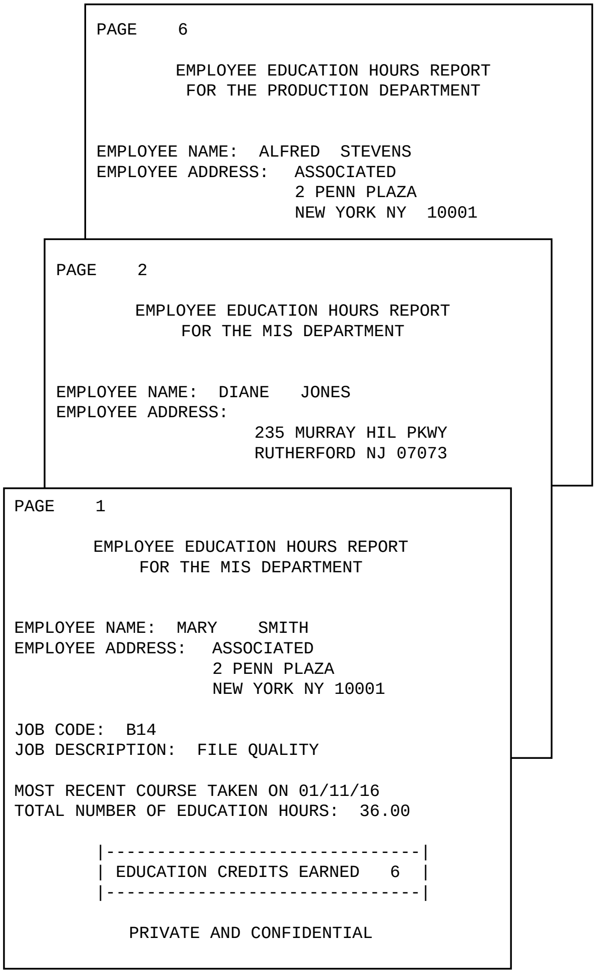
The remainder of this chapter contains the following topics:

- Analysis of a sample free-form report and the request that generated it.
- Description of how to design a free-form report using text characters, graphic characters, and data fields within headers and footers. You will also learn how to manipulate and calculate data fields for your report. See *Designing the Report* on page 18-6.
- Illustration of report sorting and record selection in a free-form report. See *Sorting the Report and Selecting Records* on page 18-8.
- Tips on structuring a report request in a stored procedure. See *Executing a Free-Form Report From a Stored Procedure* on page 18-9.

Sample Report

Suppose that you are a Personnel Manager and it is your responsibility to administer your company’s education policy. This education policy states that the number of hours of outside education that an employee can take at the company’s expense is determined by the number of hours of in-house education completed by the employee. To do your job efficiently, you would like an education hours report that shows the in-house educational history of each employee. You need each employee’s information to appear on a separate page so that it can be placed in the employee’s personnel file and referenced when an employee requests approval to take outside courses.

The Employee Education Hours Report should be formatted as follows, with a separate page for each employee. Notice that pages 1 and 2 of the report provide information about employees in the MIS department, while page 6 provides information for an employee in the Production department.



```
PAGE    6

        EMPLOYEE EDUCATION HOURS REPORT
         FOR THE PRODUCTION DEPARTMENT


EMPLOYEE NAME:  ALFRED  STEVENS
EMPLOYEE ADDRESS:   ASSOCIATED
                    2 PENN PLAZA
                    NEW YORK NY  10001
```

```
PAGE    2

        EMPLOYEE EDUCATION HOURS REPORT
             FOR THE MIS DEPARTMENT


EMPLOYEE NAME:  DIANE   JONES
EMPLOYEE ADDRESS:
                    235 MURRAY HIL PKWY
                    RUTHERFORD NJ 07073
```

```
PAGE    1

        EMPLOYEE EDUCATION HOURS REPORT
             FOR THE MIS DEPARTMENT


EMPLOYEE NAME:  MARY    SMITH
EMPLOYEE ADDRESS:   ASSOCIATED
                    2 PENN PLAZA
                    NEW YORK NY 10001

JOB CODE:  B14
JOB DESCRIPTION:  FILE QUALITY

MOST RECENT COURSE TAKEN ON 01/11/16
TOTAL NUMBER OF EDUCATION HOURS:  36.00

        |--------------------------------|
        | EDUCATION CREDITS EARNED   6  |
        |--------------------------------|

           PRIVATE AND CONFIDENTIAL
```

Sample Request

The following FOCUS request produced the Employee Education Hours Report that we saw in *Sample Report* on page 18-2. Annotations highlight different sections of this free-form report request that are explained following the sample request. Refer back to the sample report as you examine this request.

```
1.  DEFINE FILE EMPLOYEE
        CR_EARNED/I2= IF ED_HRS GE 50 THEN 9
            ELSE IF ED_HRS GE 30 THEN 6
            ELSE 3;
        END

2.  TABLE FILE EMPLOYEE
3.  HEADING
    "PAGE <TABPAGENO"
    " "
    "<13>EMPLOYEE EDUCATION HOURS REPORT"
4.  "<14>FOR THE <DEPARTMENT DEPARTMENT"
5.  "</2"
    "EMPLOYEE NAME:       <FIRST_NAME> <LAST_NAME>"
    "EMPLOYEE ADDRESS:    <ADDRESS_LN1>"
    "<19><ADDRESS_LN2>"
    "<19><ADDRESS_LN3>"
    "</1"
    "JOB CODE: <JOBCODE>"
    "JOB DESCRIPTION: <JOB_DESC>"
    "</1"
6.  "MOST RECENT COURSE TAKEN ON: <MAX.DATE_ATTEND>"
    "TOTAL NUMBER OF EDUCATION HOURS: <ED_HRS>"
    "</1"
7.  "<10>|-------------------------------|"
8.  "<10>| EDUCATION CREDITS EARNED <CR_EARNED> |"
    "<10>|-------------------------------|"
9.  FOOTING
    "<15>PRIVATE AND CONFIDENTIAL"
    BY DEPARTMENT
10. BY EMP_ID NOPRINT PAGE-BREAK
11. WHERE ED_HRS GT 0
    END
```

Explanation:

1. This request begins with a DEFINE command that enables us to create a temporary field in the report. The calculations specified in the DEFINE command reflect the company's policy for earning outside education credits. The results of this calculation are stored in CR_EARNED and appear later in the report (See note #8).

2. A free-form report begins with a standard TABLE FILE command. This sample report uses the EMPLOYEE database for its data.

3. The heading section, initiated by the HEADING phrase, defines the body of the report. Most of the text and data fields that appear on the report are specified in the heading section. In this report the heading section continues until the FOOTING phrase is encountered.

4. This request line illustrates how versatile you can be with a heading. It shows the following:

- The text to be printed in the heading.
- A data field to be embedded within the text: <DEPARTMENT.
- The heading line to begin in column 14: <14>.

5. The format of your report can be enhanced through the use of line skipping commands, such as the one we see here. Specifically for this report, we want to skip two lines between the title of the report and the name of the employee.

6. This line of the request shows how to perform field calculations in a free-form report through the use of prefix operators. In this case, we have asked to see the date on which the most recent course was taken—that is, the record's maximum value for the DATE_ATTEND field.

7. These lines illustrate the use of special characters to create graphics in your report. For our sample report, we want the calculated field and its title to appear in a box, in order to highlight the number of education credits each employee has earned.

8. This line demonstrates that you can display a defined field in the body of your report. As you recall, we defined this field at the very top of our request.

9. The FOOTING phrase not only signifies the beginning of the footing section, but, in our example, it also ends the heading section. Since we have designed a personnel report, it is important to have the words "Private and Confidential" appear at the end of each page of the report. The footing can accomplish this action.

10. This line illustrates the use of sorting in a free-form report. The report specifications require that only one employee's data appear per page; that requirement is met through the use of the BY phrase.

11. Record selection is another technique that can be employed in a free-form report. For our example, we do not want to process any employees that have not accumulated any in-house education credits, so we exclude them through the use of the WHERE phrase.

Designing the Report

To design the body of your report to your own format specifications, you need to rely on the HEADING and FOOTING commands of the FOCUS TABLE facility. These commands enable you to do the following:

- Incorporate text, data field values, and graphic characters into your report.
- Lay out your report by positioning text and data in exact column locations and skipping lines for readability.

You would use the HEADING phrase to define the body of your report and the FOOTING phrase to define what appears at the bottom of each page of your report. The use of a footer in your request is optional. You can easily define your entire report using just a header.

Specifying Text

Text can be specified anywhere in your report, for a variety of purposes. In our earlier sample report, we used text for the following purposes:

- As a report heading:

```
"<13>EMPLOYEE EDUCATION HOURS REPORT"
```

- As a label for a data field:

```
"EMPLOYEE NAME:   <FIRST_NAME> <LAST_NAME>"
```

- To specify a report page footer:

```
FOOTING
"<15>PRIVATE AND CONFIDENTIAL"
```

- In conjunction with data fields and graphic characters:

```
"<10>|EDUCATION CREDITS EARNED <CR_EARNED>|"
```

Specifying Data Fields

The crucial element in any report, free-form or otherwise, is the data displayed. The data fields that are available for use in your TABLE request include data fields in the Master File, cross-referenced fields, and temporary fields created with the DEFINE command.

The earlier sample report referenced all three types of data fields:

- ED_HRS is found in the Master File for the EMPLOYEE database:

  ```
  "TOTAL NUMBER OF EDUCATION HOURS: <ED_HRS>"
  ```

- DATE_ATTEND is found in the EDUCFILE database, which is cross-referenced to the EMPLOYEE database:

  ```
  "MOST RECENT COURSE TAKEN ON: <MAX.DATE_ATTEND>"
  ```

- CR_EARNED is defined to the EMPLOYEE database prior to issuing the TABLE FILE command:

  ```
  "<10>|EDUCATION CREDITS EARNED <CR_EARNED>|"
  ```

Using Prefix Operators

When specifying a data field on your report, you may want to use a prefix operator to specify which record to print (FST, LST, MAX, MIN) or to perform a calculation on a field value (PCT, AVE, CNT, TOT). All prefix operators can be used in a free-form report.

In the earlier sample report, we used the MAX prefix operator to print out just the most recent date on which the employee had completed an in-house course:

```
"MOST RECENT COURSE TAKEN ON: <MAX.DATE_ATTEND>"
```

As is true with all types of FOCUS reports, you must understand the structure of your database to use the prefix operators correctly.

Specifying Special Characters

The use of graphics in your report can be as creative as your imagination. In our earlier sample report we used special characters only to enclose some text and a calculated field in what appears to be a box. Some other ideas include the following:

- Highlighting key fields using asterisks or other special characters available directly from your keyboard or via the HEXBYT subroutine. (See Chapter 9, *Using Functions and Subroutines,* for details.)
- Enclosing your whole report in a box to give it a form-like appearance.
- Using double lines to separate the body of your report from its heading and footing.

The use of special characters to create graphics in your report is limited by what can be entered and viewed from your workstation and what you can print on your printer. If you have any difficulty producing the graphics that you want, be sure to check with someone in your organization who knows what is available at your location.

Report Layout

To space your report and position text and fields where you want them, you can use the positioning features of the HEADING and FOOTING phrases.

We can look at a few examples of this from the sample report in *Sample Report* on page 18-2. The first two examples show how to position both text and data fields on your report, while the third example shows how to space your report by skipping lines.

- The <13> in the following example tells FOCUS to begin printing the specified text in column 13 of the report:

  ```
  "<13>EMPLOYEE EDUCATION HOURS REPORT"
  ```

- This example tells FOCUS to begin printing the specified data field in column 19 of the report:

  ```
  "<19><ADDRESS_LN3>"
  ```

- This last example tells FOCUS to skip 1 line (embed a blank line) after printing the job description line and before printing the course name line:

  ```
  "JOB DESCRIPTION: <JOB_DESC> </1"
  "MOST RECENT COURSE TAKEN ON: <MAX.DATE_ATTEND>"
  ```

You can also manipulate your report layout using BY phrase commands such as NOPRINT, PAGE-BREAK, and UNDERLINE. An example of this appears in *Sorting the Report and Selecting Records* on page 18-8.

Sorting the Report and Selecting Records

As with columnar and matrix reports, you can both sort your report and conditionally select records for your report. You accomplish sorting and selecting for free-form reports using the same phrases you would for columnar and matrix reports:

- The BY phrase is used to sort your report.
- The WHERE phrase is used to select database records.

While designing free-form reports, you can take advantage of BY phrase commands such as NOPRINT, PAGE-BREAK, and UNDERLINE. In our earlier sample report, we used this technique to place each employee’s information on a separate page.

```
BY EMP_ID NOPRINT PAGE-BREAK
```

Executing a Free-Form Report From a Stored Procedure

If you plan to execute your free-form report on a regular basis, you will probably want to create and then store the procedure. The following reminders will help you to structure your stored procedure properly.

- Create any temporary fields in your report with a DEFINE command prior to initiating the TABLE FILE command. Do not forget to complete your DEFINE request with an END command.
- Start your report request with a TABLE FILE command.
- Define your header specifications.
- Define your footer specifications, if needed.
- Specify any sorting requirements with the BY phrase.
- Specify any selection criteria with the WHERE phrase.
- End your report request with an END command.

19 Creating Graphs: GRAPH

Topics:

- Introduction
- Command Syntax
- Graph Forms
- Adjusting Graph Elements
- Special Topics
- Special Graphics Devices
- Command and SET Parameter Summary

Graphs often convey meanings more clearly than data listed in tabular report format. The FOCUS GRAPH command acts in the same way as the TABLE command to retrieve data from a file, but it presents the information—either on the screen or to a printer in one of five standard graphic formats:

- A connected point or line plot
- A histogram
- A bar chart
- A pie chart
- A scatter diagram

Introduction

This chapter explains how to generate each of these graph forms and adjust the features on the graphs you produce.

The examples in this chapter are drawn on the SALES database which is included on your system tape. All examples in this chapter assume that FOCUS default parameters, called SET parameters, are in effect.

The SALES database is used to illustrate the examples used in this chapter. The Master File and a schematic diagram of the file appear in Appendix A, *Master Files and Diagrams*. An additional temporary field named SALES has been defined and used in many of the examples:

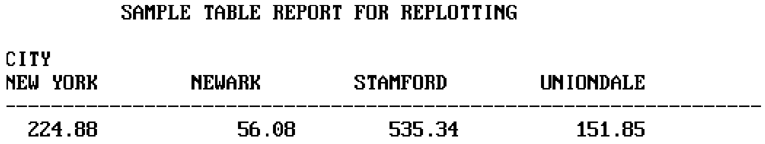
```
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END
```

GRAPH vs. TABLE Requests

GRAPH request syntax is similar to TABLE request syntax. In fact, the output from many TABLE requests can be converted directly into a graph by typing the command REPLOT at the FOCUS command prompt immediately after the output of the request has been displayed. For example

```
TABLE FILE SALES
HEADING CENTER
"SAMPLE TABLE REPORT FOR REPLOTTING"
SUM SALES ACROSS CITY
END
```

produces the following output:

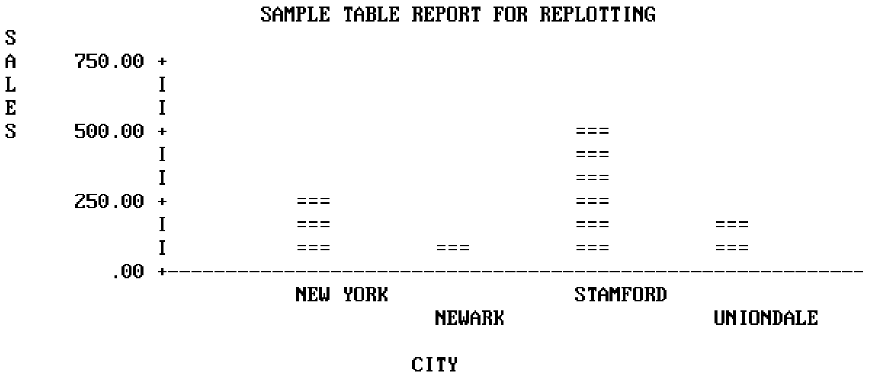


```
          SAMPLE TABLE REPORT FOR REPLOTTING

CITY
NEW YORK        NEWARK        STAMFORD        UNIONDALE
-----------------------------------------------------------------
  224.88            56.08        535.34          151.85
```

To convert the output into a graph, exit the report and at the FOCUS command prompt type

```
REPLOT
```

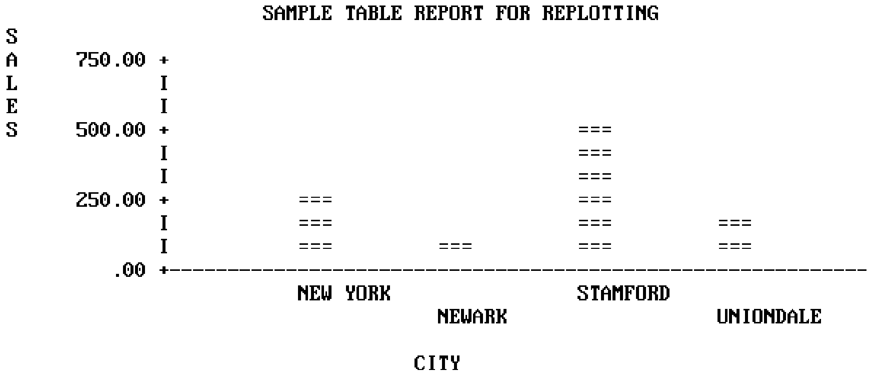
and press Enter.



```
                      SAMPLE TABLE REPORT FOR REPLOTTING
S
A     750.00 +
L            I
E            I
S     500.00 +                                   ===
             I                                   ===
             I                                   ===
      250.00 +           ===                     ===
             I           ===                     ===         ===
             I           ===         ===         ===         ===
         .00 +------------------------------------------------------------
                         NEW YORK                STAMFORD
                                     NEWARK                  UNIONDALE

                                   CITY
```

To produce the graph directly, without creating a preliminary tabular report, substitute the command GRAPH for TABLE in the original request, as shown in the following example:

```
GRAPH FILE SALES
HEADING CENTER
"SAMPLE TABLE REPORT FOR REPLOTTING"
SUM SALES ACROSS CITY
END
```

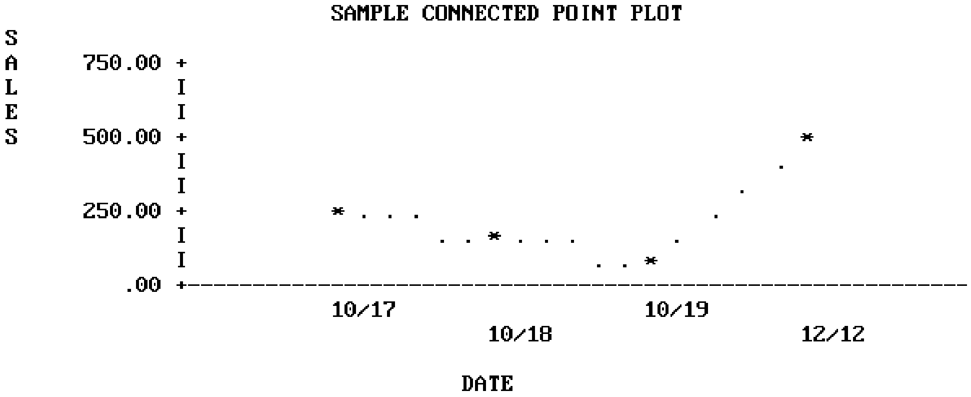


```
                      SAMPLE TABLE REPORT FOR REPLOTTING
S
A     750.00 +
L            I
E            I
S     500.00 +                                   ===
             I                                   ===
             I                                   ===
      250.00 +           ===                     ===
             I           ===                     ===         ===
             I           ===         ===         ===         ===
         .00 +------------------------------------------------------------
                         NEW YORK                STAMFORD
                                     NEWARK                  UNIONDALE

                                   CITY
```

Thus, you can produce graphs by simply converting TABLE requests, but every TABLE facility does not have a GRAPH counterpart, and there are some practical limitations on the amount of information that you can effectively display in a graph. *Command Syntax* on page 19-13 describes the use of TABLE features in GRAPH requests.

The type of graph (graph form) produced by a GRAPH request depends on the verb used (such as SUM or PRINT), the sort phrase used (ACROSS or BY), and the data type of the sort field. Consider the five graphs that appear on the following pages, and the requests that produce them.

```
SET HISTOGRAM=OFF

GRAPH FILE SALES
HEADING CENTER
"SAMPLE CONNECTED POINT PLOT"
SUM SALES ACROSS DATE
END
```

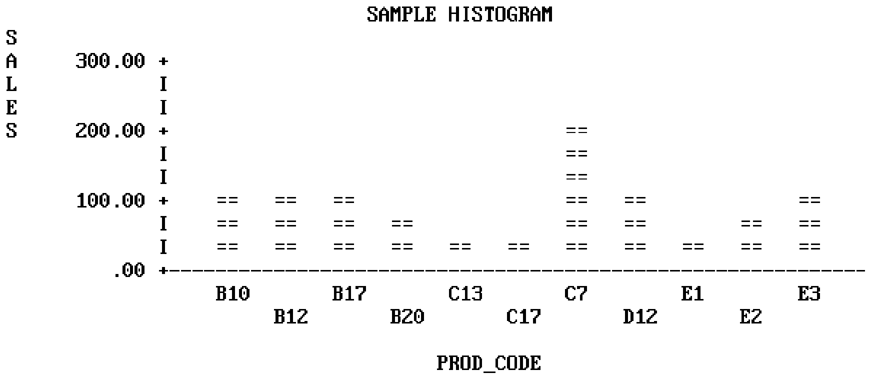


```
                         SAMPLE CONNECTED POINT PLOT
S
A     750.00 +
L            I
E            I
S     500.00 +                                               *
             I                                             .
             I                                          .
      250.00 +           * . . .                      .
             I                   . . * . . .       .
             I                               . . *
         .00 +------------------------------------------------------------
                         10/17                   10/19
                                     10/18                   12/12

                                   DATE
```

Figure 19-1. Sample Connected Point Plot

Note: SET parameters remain in effect until you reset them or log off (see *SET Parameters* on page 19-59).

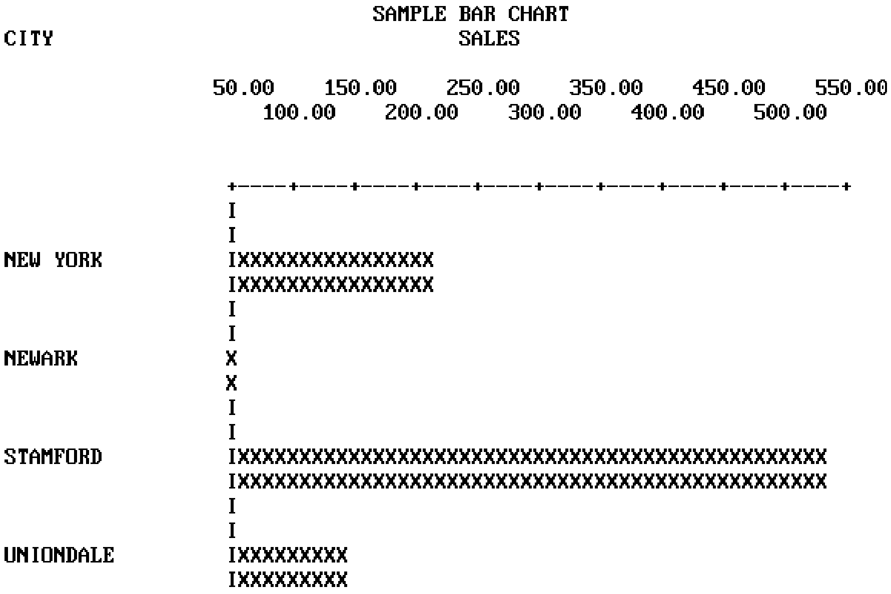
```
SET HISTOGRAM=ON

GRAPH FILE SALES
HEADING CENTER
"SAMPLE HISTOGRAM"
SUM SALES ACROSS PROD_CODE
END
```



```
                               SAMPLE HISTOGRAM
S
A     300.00 +
L            I
E            I
S     200.00 +                                  ==
             I                                  ==
             I                                  ==
      100.00 +    ==   ==   ==                  ==   ==             ==
             I    ==   ==   ==   ==             ==   ==        ==   ==
             I    ==   ==   ==   ==   ==   ==   ==   ==   ==   ==   ==
         .00 +------------------------------------------------------------
                  B10       B17       C13       C7        E1        E3
                       B12       B20       C17       D12       E2

                                     PROD_CODE
```

Figure 19-2. Sample Histogram

```
SET BARWIDTH=2, BARSPACE=2

GRAPH FILE SALES
HEADING CENTER
"SAMPLE BAR CHART"
SUM SALES BY CITY
END
```



```
                              SAMPLE BAR CHART
CITY                                 SALES

                 50.00    150.00    250.00    350.00    450.00    550.00
                     100.00    200.00    300.00    400.00    500.00


                  +----+----+----+----+----+----+----+----+----+----+
                  I
                  I
NEW YORK          IXXXXXXXXXXXXXXXX
                  IXXXXXXXXXXXXXXXX
                  I
                  I
NEWARK            X
                  X
                  I
                  I
STAMFORD          IXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                  IXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                  I
                  I
UNIONDALE         IXXXXXXXXX
                  IXXXXXXXXX
```

Figure 19-3. Sample Bar Chart

```
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END

SET PIE=ON, GCOLOR=OFF
SET VAXIS=50, HAXIS=100

GRAPH FILE SALES
HEADING CENTER
"SAMPLE PIE CHART"
SUM SALES ACROSS CITY
END
```

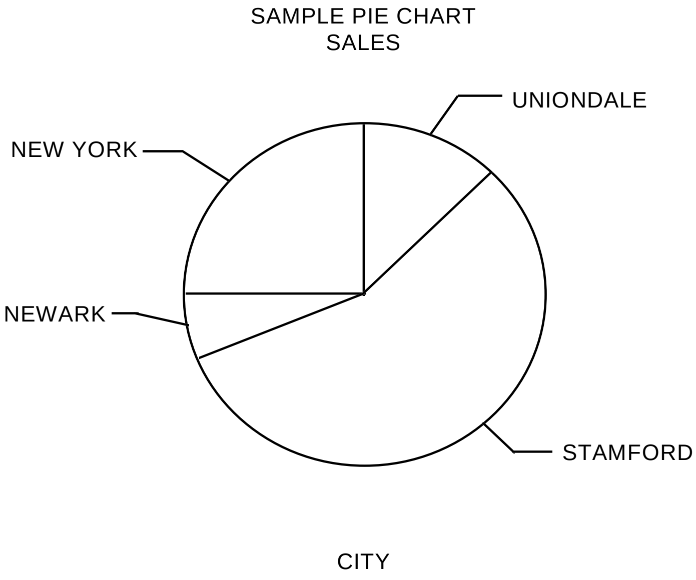
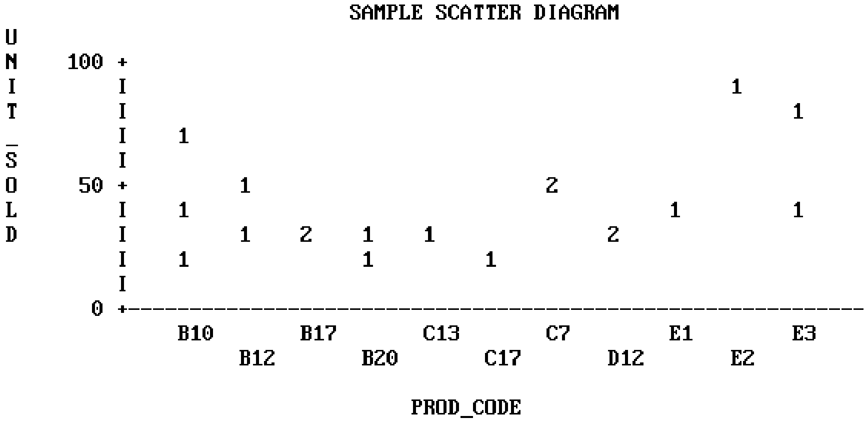


Figure 19-4. Sample Pie Chart

```
SET HISTOGRAM=OFF

GRAPH FILE SALES
HEADING CENTER
"SAMPLE SCATTER DIAGRAM"
PRINT UNIT_SOLD ACROSS PROD_CODE
END
```



```
                            SAMPLE SCATTER DIAGRAM
U
N    100 +
I        I                                                 1
T        I                                                      1
_        I    1
S        I
O     50 +         1                        2
L        I    1                                       1         1
D        I         1    2    1    1              2
         I    1              1         1
         I
       0 +------------------------------------------------------------
              B10       B17       C13       C7        E1        E3
                   B12       B20       C17       D12       E2

                                 PROD_CODE
```

Figure 19-5. Sample Scatter Diagram

Controlling the Format

In each of the previous graphs, FOCUS created a clear representation of the data using its default values for the graph features (such as axis lengths, axis scales, or titles). You can issue your initial request immediately and concentrate on selecting the data, while FOCUS controls all of the features on the graph.

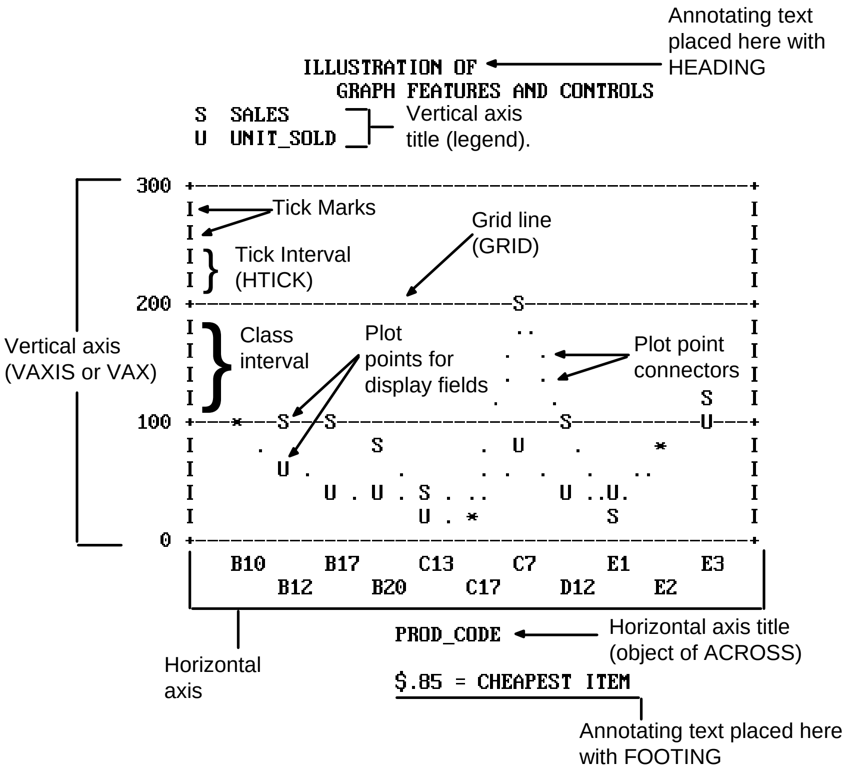
Subsequently, when satisfied with the data portrayed in your graph, you can refine its appearance by adjusting the parameters that control the look of the graph. You can set the control parameters individually (for example, SET GRID=ON), or ask FOCUS to prompt you for all of their values when you execute the SET GPROMPT=ON command.

Note: When entering several SET parameters on one line, separate them with commas.

The request below illustrates some of the parameters you can control:

```
SET HISTOGRAM=OFF
SET HAXIS=75, VAXIS=32, GRID=ON

GRAPH FILE SALES
HEADING
"</1 <20 ILLUSTRATION OF"
"<23 GRAPH FEATURES AND CONTROLS"
SUM SALES AND UNIT_SOLD ACROSS PROD_CODE
FOOTING CENTER
"</1 <MIN.RETAIL = CHEAPEST ITEM"
END
```

The graph generated OFFLINE in response to the request appears below.



```
              ILLUSTRATION OF
                GRAPH FEATURES AND CONTROLS
    S  SALES
    U  UNIT_SOLD

300 +-----------------------------------------------+
    I                                               I
    I                                               I
    I                                               I
    I                                               I
200 +---------------------------S-------------------+
    I                           ..                  I
    I                          .  .                 I
    I                          .  .                 I
    I                         .    .            S   I
100 +---*---S---S-------------------S-----------U---+
    I     .         S        .  U    .      *       I
    I       U .       .      . .  .   .   ..        I
    I           U . U . S . ..      U ..U.          I
    I                   U . *           S           I
  0 +-----------------------------------------------+
       B10     B17     C13     C7      E1      E3
           B12     B20     C17     D12     E2

                     PROD_CODE

                     $.85 = CHEAPEST ITEM
```

Annotating text placed here with HEADING
Vertical axis title (legend).
Tick Marks
Grid line (GRID)
Tick Interval (HTICK)
Vertical axis (VAXIS or VAX)
Class interval
Plot points for display fields
Plot point connectors
Horizontal axis
Horizontal axis title (object of ACROSS)
Annotating text placed here with FOOTING

Figure 19-6. Illustration of Graph Elements

Note:

- This graph is a connected point plot, with the plot points representing the sales (retail_price * unit_sold) and total units sold for each of the product codes listed across the horizontal axis.
- Annotating text has been added above and below the graph with the HEADING and FOOTING facilities. Note the use of spot markers to position text on the graph and the embedded calculation with a direct operator.
- Only the vertical axis is scaled because the ACROSS phrase objects were not numeric values. The plus symbols (+) mark the class intervals on the axis scale and vertical bars mark the tick intervals.
- Horizontal grid lines appear at the vertical class marks.

For graphs generated ONLINE, FOCUS automatically detects the height and width of a particular terminal and plots the graph accordingly. As a result, VAXIS and HAXIS settings are ignored.

You control the graphic elements shown in Figure 19-6 in one of two ways: either by the syntax in the actual request, or with SET commands. *Command Syntax* on page 19-13 describes the elements in GRAPH requests and their effects. *Adjusting Graph Elements* on page 19-38 describes the adjustable parameters that control graph features.

There are also some additional SET parameters that control non-graphic elements:

- Specifying an output device.
- Pausing between data retrieval and printing to permit the user to adjust paper in the printer or plotter.
- Using special black/white shading patterns to simulate different colors.
- Displaying the current settings of the GRAPH parameters on the screen.

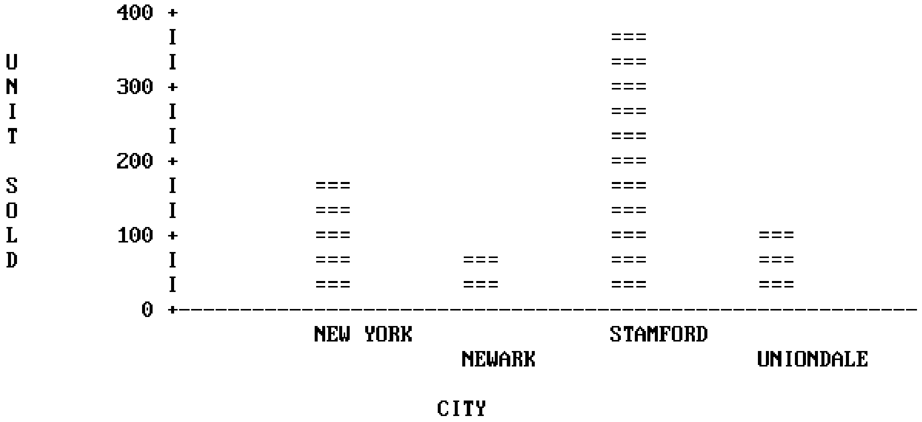
After retrieving data from a file and displaying it either as a tabular report or a graph, you can use the SET command to adjust the format and then redisplay the graph by issuing the REPLOT command (without resorting to further data retrieval).

A summary of all of the SET parameters appears in *SET Parameters* on page 19-59.

Graphic Devices Supported

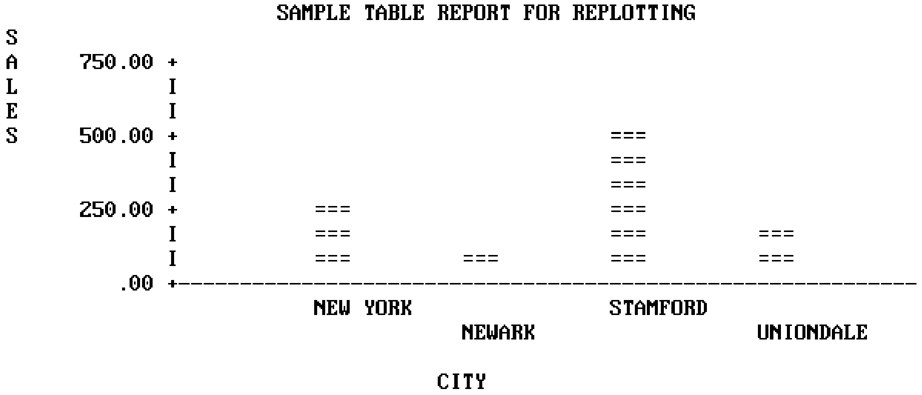
You may create graphs on any terminal or printer that can print FOCUS reports. If your terminal has no graphics capabilities, FOCUS uses the characters in the standard character set when producing graphs. As the default, FOCUS sends GRAPH output to the terminal (or system printer, if PRINT=OFFLINE). This produces low-resolution graphics. The examples in the chapter thus far (except the pie chart) illustrate the default. You cannot create continuous line plots or pie charts unless you have a high-resolution graphic device.

While FOCUS can accommodate devices with no inherent graphics capabilities, it can also take advantage of whatever graphics facilities are available. Some personal computers offer ranges of special characters that can be used to create more readable graphs (see Figure 19-7), and if color monitors or multiple-pen plotters are available, further improvements in graph quality are possible (see Figure 19-8 and *Special Graphics Devices* on page 19-53).



```
         400 +
             I                                   ===
U            I                                   ===
N        300 +                                   ===
I            I                                   ===
T            I                                   ===
         200 +                                   ===
S            I           ===                     ===
O            I           ===                     ===
L        100 +           ===                     ===         ===
D            I           ===         ===         ===         ===
             I           ===         ===         ===         ===
           0 +------------------------------------------------------------
                         NEW YORK                STAMFORD
                                     NEWARK                  UNIONDALE

                                   CITY
```

Figure 19-7. Graph on an IBM PC Mono Screen



```
                      SAMPLE TABLE REPORT FOR REPLOTTING
S
A     750.00 +
L            I
E            I
S     500.00 +                                   ===
             I                                   ===
             I                                   ===
      250.00 +           ===                     ===
             I           ===                     ===         ===
             I           ===         ===         ===         ===
         .00 +------------------------------------------------------------
                         NEW YORK                STAMFORD
                                     NEWARK                  UNIONDALE

                                   CITY
```

Figure 19-8. Sample Graph on Plotter

On IBM mainframes, FOCUS supports the use of high-resolution terminals such as the Model 3279 via the IBM Graphical Data Display Manager (GDDM), which is discussed in *IBM Devices Using GDDM* on page 19-54. A variety of other high-resolution terminals, printers, and plotters are also supported and they are listed in this section. To select one, simply enter the appropriate form of the SET DEVICE command (see *High-Resolution Devices* on page 19-54). Note, in reviewing the device selections, that all have fixed graphic window dimensions (horizontal and vertical axes), which are fixed until a new device is selected.

Please note that this list includes only fully tested devices, although other devices may also work with FOCUS.

Medium-Resolution Devices

Anderson Jacobson Models:AJ830, and AJ832 (12 Pitch).

Diablo Models: 1620, and 1620 (12 Pitch).

Gencom Models: GENCOM, and GENCOM (12 Pitch).

Trendata Models: Trendata 4000A, and 4000A (12 Pitch).

High-Resolution Devices

IBM Graphic Devices (GDDM is required).

- Any IBM 3270 series device that supports GDDM graphics, such as 3279-S3G, 3179, or 3472. This includes PCs with fully compatible 3270 series hardware and software.

Hewlett-Packard Plotters:

- Four-pen plotters without paper advance: Models 7220A, 7221, and 7470A (requires Y cable #17455).
- Four-pen plotters with paper advance: Models 7220S and 7221S.
- Eight-pen plotters without paper advance: Models 7220C, 7221C, 7475A (requires Y cable #17455).
- Eight-pen plotters with paper advance: Models 7220T, 7221T.

Tektronix Graphic Devices (only monochrome display).

- Models 4010, 4012, 4013, 4014, 4014E, 4015, 4015E, 4025, 4027, 4050 series, 4662, and 4100 series.

Command Syntax

Most TABLE requests can be converted into GRAPH requests by simply replacing the TABLE command with the GRAPH command. The only limitations are those inherent in the nature of the graphic format. When a TABLE request is converted in this manner, the various phrases that make up the body of the request take on special meanings that determine the format and layout of the graph.

This section outlines the phrases that can appear in TABLE requests and describes their effects in the context of GRAPH requests. The section also describes any limitations that apply to their use.

GRAPH vs. TABLE Syntax

The syntax of the GRAPH command parallels that of the TABLE command. The main elements of GRAPH requests are the verb phrase (display command), one or more sort phrases, selection phrases, and headings and footings. All of the other phrases that appear in TABLE requests are ignored. This applies to all control conditions (ON…) and all IN phrases. The basic GRAPH syntax is as follows:

```
GRAPH FILE filename
[HEADING]
[heading phrase]
verb phrase
sort phrase
[additional sort phrases]
[selection phrase(s)]
[FOOTING]
[footing phrase]
END
```

The GRAPH request elements generally follow the same rules as their TABLE counterparts:

- The word FILE and the file name must immediately follow the GRAPH command, unless they were previously specified in a SET command:

  ```
  SET FILE=filename
  ```

  The file named can be any file available to FOCUS, including joined or cross-referenced structures.

- You can concatenate unlike data sources in a GRAPH request with the MORE command. See *Concatenating Unlike Data Sources* on page 19-18.

- The order of the phrases in the request does not affect the format of the graph. For example, the selection phrase may follow or precede the verb phrase and sort phrase(s). The order of the sort phrases does affect the format of the graph, however, just as the order of the sort phrases in TABLE requests affects the appearance of the reports (see *Selecting Forms: BY and ACROSS Phrases* on page 19-16).
- The word END must be typed on a line by itself to complete a GRAPH request.
- An incomplete GRAPH request can be terminated by typing the word QUIT on a line by itself (instead of END).
- All dates are displayed in MDY format unless they are changed to alphanumeric fields.

There are a few notable syntactical differences between TABLE and GRAPH. Specifically, the following restrictions apply:

- GRAPH requests must contain at least one sort phrase (BY phrase or ACROSS phrase) and a verb with at least one verb object in order to generate a meaningful graph.
- Several BY phrases can be used in a request, in which case multiple graphs are created (one for each BY object). A single ACROSS phrase is allowed in a GRAPH request, and requests for certain graph forms can contain both ACROSS and BY phrases.
- The number of ACROSS values cannot exceed 64.
- In GRAPH requests the verb object must always be a numeric field.
- In GRAPH requests the sortfield in ACROSS and BY phrases cannot be a date format field.
- No more than five verb objects are permitted in a GRAPH request. (This limitation is necessary because standard graph formats generally do not permit more variables to be displayed without rendering the graph unreadable.)
- The RUN option is not available as an alternative to END.

The following sections describe in more detail the functions performed by each of the phrases used in GRAPH requests.

Specifying Graph Forms and Contents

Each graph form is defined by a particular combination of verb and sort phrase. The combinations, which were illustrated earlier in *GRAPH vs. TABLE Requests* on page 19-2, are summarized in the table below (A and B represent two field names).

```
Point plot:SUM A ACROSS B (B is numeric)
Histogram: SUM A ACROSS B (B is alpha) requires SET HISTOGRAM=ON
Bar chart: SUM A BY B
Pie chart: SET PIE=ON
           SUM A ACROSS B

Scatter diagram:      PRINT A ACROSS B or PRINT A BY B
```

Figure 19-9. Graph Selection Phrases

Naming Subjects: Verb Phrases

Each GRAPH request must include a verb and at least one verb object (up to five are allowed). Three verbs are permitted: COUNT, SUM, and PRINT. SUM is synonymous with either WRITE or ADD. Each verb object must be a computational field (not alphabetic). For example,

```
GRAPH FILE SALES
SUM SALES
.
.
.
```

If the verb SUM (or WRITE or ADD) is used, then a bar chart, histogram, line plot or pie chart is produced, depending on the sort phrase and sort field format used. If PRINT is used, the graph is a scatter diagram.

The verb objects, which are the subjects of the graph, may be real or defined fields, with or without direct operation prefixes (AVE., MIN., MAX., etc.). They may also be computed fields. (All of the COMPUTE facilities are available in GRAPH requests.)

When the request has a single verb object, the vertical title of the graph is either the field name of the verb object as it appears in the Master File or a replacement name supplied in an AS phrase.

When a request contains multiple verb objects, each represents one variable in the graph, and a vertical legend is printed instead of the vertical title. The legend specifies the field names (and/or AS phrase substitutions) and provides a key to which line represents each variable.

In your requests, verb objects may be separated by spaces, or by AND or OVER. OVER has special significance in histogram and bar chart requests, where it controls the stacking of the bars. This is described in the sections on Histograms (see *Histograms* on page 19-26), and Bar Charts (see *Bar Charts* on page 19-29).

Verb objects used only for calculations need not appear in your graphs. Use the NOPRINT or SUP-PRINT facilities to suppress the display of such fields.

Selecting Forms: BY and ACROSS Phrases

At least one sort phrase is required in every GRAPH request. This may be either a BY phrase or an ACROSS phrase.

For example,

```
GRAPH FILE SALES
SUM SALES
ACROSS PROD_CODE
.
.
.
```

The ACROSS phrase, if there is one, determines the horizontal axis of the graph.

If there is no ACROSS phrase the last BY phrase determines the vertical axis. When there are multiple BY phrases or when an ACROSS and BY phrase are included in the same request, FOCUS generates multiple graphs; one for each combination of values for the fields referenced in the request (see *The Vertical Axis: System Defaults* on page 19-42 for information regarding control of the vertical axis).

Note: The FOCUS ICU Interface saves data for IBM's Interactive Chart Utility (ICU) in tied data format. If both an ACROSS and BY phrase are present in a GRAPH request one common axis is established. This enables FOCUS graphs to be displayed as tower charts.

The FOCUS ICU Interface is discussed in further detail in *Using the FOCUS ICU Interface* on page 19-52. You can also consult the *ICU Interface Users Manual* for additional information.

The sortfield name may be replaced with an AS phrase. This is useful if the sort phrase specifies one of the axes (it has no effect on any additional sort phrases).

Note that the values of fields mentioned in the additional sort phrases are not displayed automatically in the graph. If you wish to have them appear you must embed them in a heading or a footing line (see *Adding Annotating Text: HEADING and FOOTING Lines* on page 19-19).

Selecting the Contents: Selection Phrases

Selection phrases are used in GRAPH requests to select particular records of interest. Two selection phrases are available: IF and WHERE. The examples in this chapter use the IF selection phrase. For a definition of the WHERE clause and the differences between IF and WHERE, see Chapter 5, *Selecting Records for Your Report*.

The syntax for an IF phrase or a WHERE clause in a GRAPH request is identical to that used in a TABLE request. For example,

```
GRAPH FILE SALES
SUM SALES
ACROSS PROD_CODE
IF PROD_CODE NE D12
```

A partial list of the relation tests appears below. See Chapter 5, *Selecting Records for Your Report*, for a complete list.

Relation	**Meaning**
`EQ`	Equal to
`NE`	Not equal to
`GE`	Greater than or equal to
`GT`	Greater than
`LE`	Less than or equal to
`LT`	Less than
`CONTAINS`	Contains
`OMITS`	Omits

Concatenating Unlike Data Sources

With the FOCUS command, MORE, you can graph data from unlike data sources in a single request; all data, regardless of source, appears to come from a single file. You must divide your request into:

- One main request that retrieves the first file and defines the data fields, sorting criteria, and output format for all data.
- Subrequests that define the files and fields to be concatenated to the data of the main request. The fields printed and sorted by the main request must exist in each concatenated file. If they do not, you must create them as DEFINE fields.

During retrieval, FOCUS gathers data from each database in turn. It then sorts all data and formats the output as described in the main request. The syntax is

```
GRAPH FILE file1
   main request
MORE
FILE file2
   subrequest
 MORE
      .
      .
      .
END
```

where:

file1

Is the name of the first file.

main request

Is a request, without END, that describes the sorting, formatting, aggregation, and COMPUTE field definitions for all data. IF and WHERE phrases in the main request apply only to *file1*.

MORE

Begins a subrequest. The number of subrequests is limited only by available memory.

FILE *file2*

Defines *file2* as the second file for concatenation.

subrequest

Is a subrequest. Subrequests can only include WHERE and IF phrases.

END

Ends the request.

See Chapter 15, *Merging Data Files*, for complete information and for concatenation examples.

Adding Annotating Text: HEADING and FOOTING Lines

To insert annotating text above or below a graph, enter the keywords HEADING and/or FOOTING, followed by the desired contents, including any necessary control elements for skipping lines, etc. The syntax is the same as that used for headings and footings in TABLE requests.

For example,

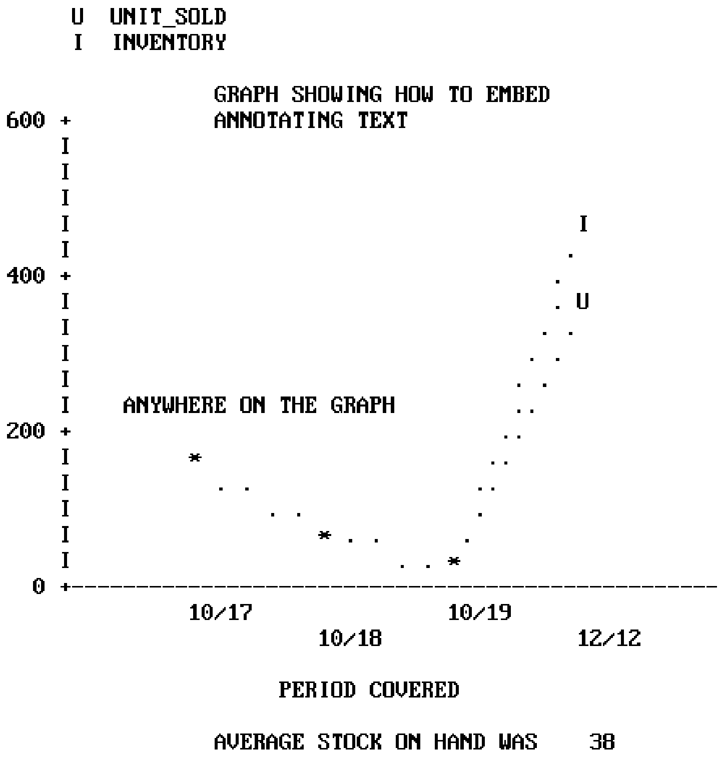
```
GRAPH FILE SALES
HEADING
"<7 THIS GRAPH SHOWS SALES BY PRODUCT CODE"
SUM SALES
BY PROD_CODE
IF PROD_CODE NE D12
FOOTING
"<7 FOR ALL PRODUCT CODES EXCEPT D12"
END
```

Note: When annotating text falls in the path of a plot point on a graph, the plot point is printed; however, connecting points are suppressed if they lie in the path of annotating text. (This enables you to adjust the position of the annotating text when you see the contents of the graph.) The first line of any heading appears above the first line of the legend.

Inserting Formatting Controls

The formatting controls used in TABLE requests can also be used in GRAPH requests for positioning text or field references in heading or footing lines, or even in the body of your graph. The following example shows the use of spot markers, which are described in Chapter 10, *Customizing Tabular Reports*.

```
SET HISTOGRAM=OFF
GRAPH FILE SALES
HEADING
"</4 <22 GRAPH SHOWING HOW TO EMBED"
"<22 ANNOTATING TEXT"
"</10 <15 ANYWHERE ON THE GRAPH"
SUM UNIT_SOLD AND OPENING_AMT AS 'INVENTORY'
ACROSS DATE AS '              PERIOD COVERED'
FOOTING CENTER
"AVERAGE STOCK ON HAND WAS <AVE.OPENING_AMT"
END
```

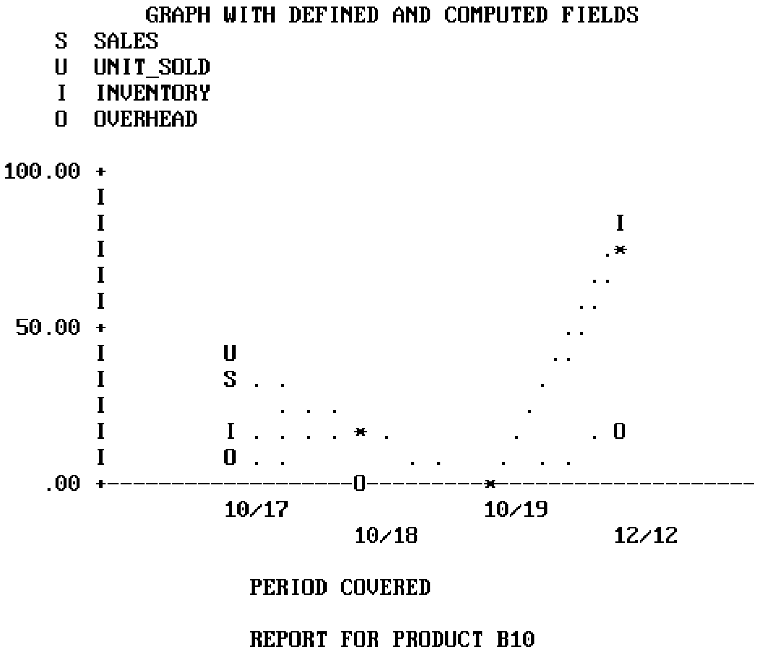


```
      U  UNIT_SOLD
      I  INVENTORY

                 GRAPH SHOWING HOW TO EMBED
 600 +           ANNOTATING TEXT
     I
     I
     I
     I                                       I
     I                                      .
 400 +                                     .
     I                                     . U
     I                                    . .
     I                                   . .
     I                                  . .
     I    ANYWHERE ON THE GRAPH         ..
 200 +                                 ..
     I         *                      ..
     I           . .                 ..
     I               . .             .
     I                   * . .      .
     I                         . . *
   0 +--------------------------------------------------
               10/17               10/19
                         10/18               12/12

                      PERIOD COVERED

                 AVERAGE STOCK ON HAND WAS    38
```

Inserting Field References

The following example shows how to embed field values in graph heading or footing lines. This capability is analogous to that in TABLE requests, and is useful when annotating graphs created by requests containing multiple sort fields (where only the first named sort field will appear as a title on the graph).

```
SET HISTOGRAM=OFF
DEFINE FILE SALES
SALES/D8.2=RETAIL_PRICE * UNIT_SOLD;
END

GRAPH FILE SALES
HEADING CENTER
"GRAPH WITH DEFINED AND COMPUTED FIELDS"
SUM SALES AND UNIT_SOLD AND OPENING_AMT
AS 'INVENTORY' AND
COMPUTE OVERHEAD/D8.2=.20 * SALES;
ACROSS DATE AS '  PERIOD COVERED'
BY PROD_CODE
IF 'PROD_CODE' IS 'C7' OR 'B10' OR 'B12'
FOOTING CENTER
"REPORT FOR PRODUCT <PROD_CODE"
END
```



```
           GRAPH WITH DEFINED AND COMPUTED FIELDS
    S  SALES
    U  UNIT_SOLD
    I  INVENTORY
    O  OVERHEAD

100.00 +
       I
       I                                       I
       I                                      .*
       I                                     ..
       I                                    ..
 50.00 +                                   ..
       I         U                        ..
       I         S . .                   .
       I             . . .              .
       I         I . . . . * .         .     . O
       I         O . .         . .    .  . .
   .00 +-------------------O---------*--------------------
                 10/17               10/19
                           10/18               12/12

                   PERIOD COVERED

                   REPORT FOR PRODUCT B10
```

Graph Forms

This section describes the five graph forms produced by FOCUS, and their basic elements. Connected point plots are described first, followed by histograms, bar charts, pie charts, and scatter diagrams. The adjustable graphic features are mentioned only briefly with the graph forms and fully described in *Adjusting Graph Elements* on page 19-38.

As you may have noticed when viewing the examples in *GRAPH vs. TABLE Requests* on page 19-2, there are similarities between the requests used to create some of the forms. For example, a request for a connected point plot (with an alphanumeric ACROSS field) will create a histogram instead if the HISTOGRAM parameter is set on (the default). This feature enables you retrieve data once, and then flip back and forth from one form to the other by changing the HISTOGRAM value and issuing REPLOT.

Histograms are often called vertical bar charts, but the physical similarities between these forms mislead users. Although the graphs look similar and have parameters that perform similar functions (HSTACK and BSTACK), the parameters used to control the widths and spacing of bars on bar charts have no effect on histogram bars.

Histograms and vertical scatter plots (those created with BY phrases) have variable-length vertical axes that are not subject to the VAXIS parameter setting (VAXIS is ignored).

Pie charts and bar charts are different geometrical representations of similar types of data, but pie charts are only possible if you have a high-resolution device capable of drawing respectable curves.

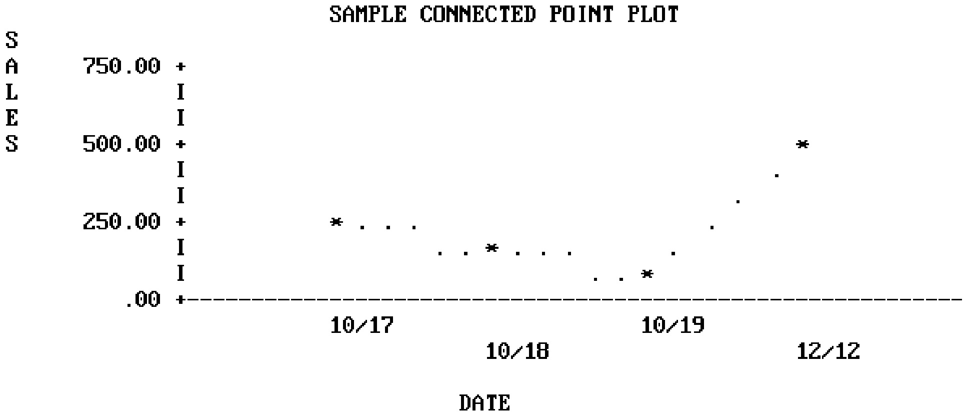
Connected Point Plots

You create a connected point plot (or a line plot on a high-resolution device), with a request that combines the verb SUM (or the synonyms WRITE or ADD) with an ACROSS phrase that specifies an alphanumeric or a numeric field. If the field specified in the ACROSS phrase is alphanumeric, the HISTOGRAM parameter must be set off in order to generate a connected point plot.

The values for the field named in the ACROSS phrase are plotted on the horizontal axis and the values for the verb object(s) are plotted along the vertical axis.

The example below illustrates a point plot request.

```
SET HISTOGRAM=OFF

SET VAXIS=40,HAXIS=75
GRAPH FILE SALES
HEADING CENTER
"SAMPLE CONNECTED POINT PLOT"
SUM SALES ACROSS DATE
END
```



```
                         SAMPLE CONNECTED POINT PLOT
S
A     750.00 +
L            I
E            I
S     500.00 +                                               *
             I                                             .
             I                                          .
      250.00 +           * . . .                      .
             I                   . . * . . .       .
             I                               . . *
         .00 +------------------------------------------------------------
                         10/17                   10/19
                                     10/18                   12/12

                                   DATE
```

Note: The SET statements in the previous example were added to limit the size of the output graph to a convenient size for display on the page. Without them, FOCUS sets the default horizontal axis width at the capacity of the device selected, and a vertical height of 66 lines (normal page length).

Point Plot Features

Scale Titles — The values associated with the class markers are printed below the horizontal axis in the USAGE format of the variable being plotted (MM/DD in our example).

Plot Characters — The graphics characters used to plot the variables on connected point plots vary depending on the type of display device being used:

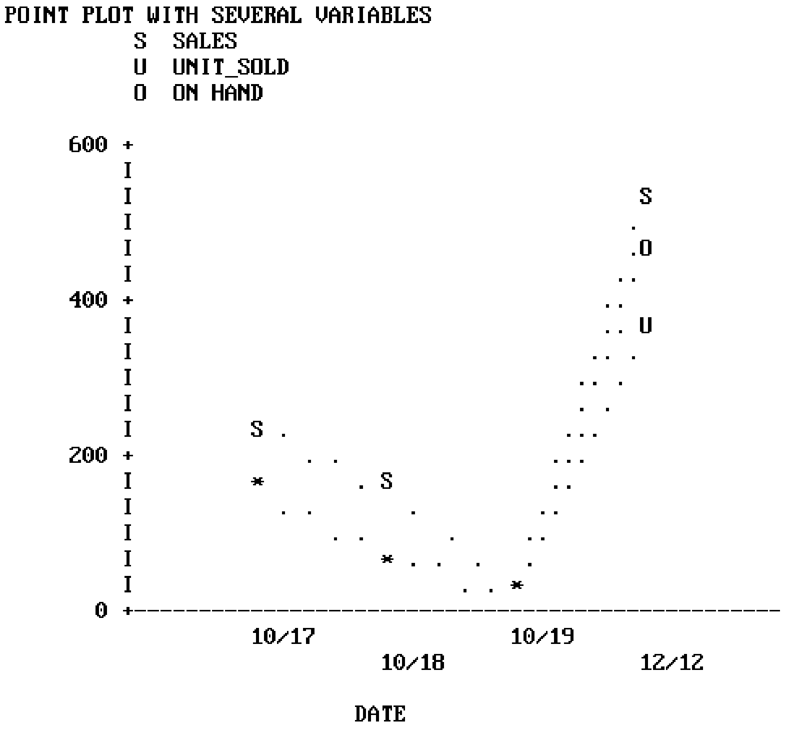
- On high-speed printers and non-graphics terminals the data points are represented by asterisks (*) when only one variable is plotted. If several variables are plotted, the initial letters of the variable names are used (if you have duplicates, rename them with AS phrases). The data points are connected by periods(.). You cannot create continuous line plots on these devices; they are only available on high-resolution devices.
- On high-resolution displays, printers, and plotters the lines connecting plot points are drawn explicitly and when there are several variables they are distinguished either by color or by the type of connecting line used (dotted, solid, or broken).

Axis Titles — You can include vertical and horizontal axis titles for your graphs:

- For requests with a single verb object the vertical title is either its field name or a replacement name you have provided in an AS phrase.
- When more than one variable is plotted, FOCUS prints a vertical legend instead of the vertical title. The legend specifies the field names or their replacements, and provides a key showing which line represents each variable. Titles are displayed staggered or folded on successive horizontal lines to permit more titles than a single horizontal line can contain.

The example that follows illustrates a point plot with several variables.

```
SET HISTOGRAM=OFF

GRAPH FILE SALES
HEADING
"POINT PLOT WITH SEVERAL VARIABLES"
SUM SALES AND UNIT_SOLD AND INV AS 'ON HAND'
ACROSS DATE
END
```

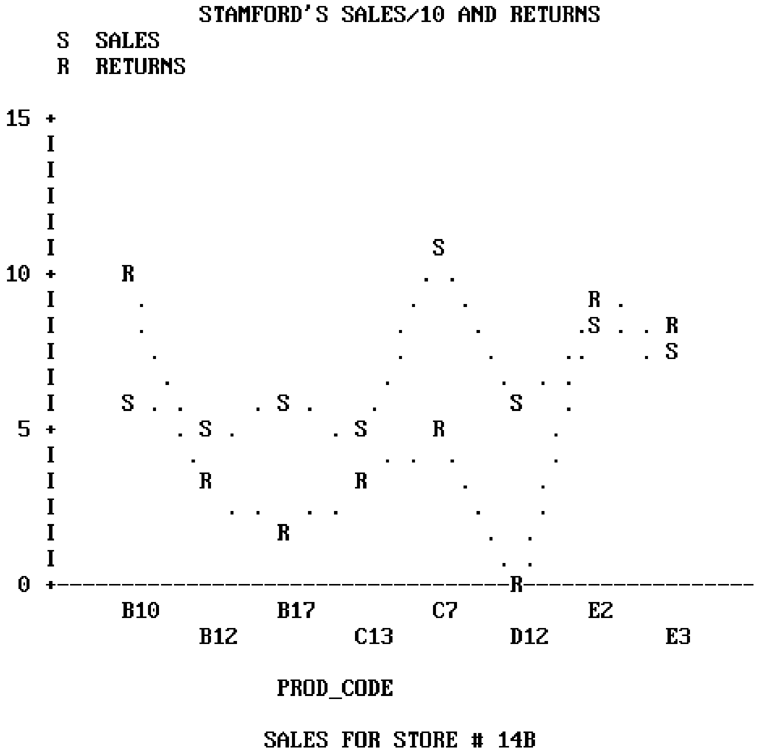


```
POINT PLOT WITH SEVERAL VARIABLES
          S  SALES
          U  UNIT_SOLD
          O  ON HAND

     600 +
         I
         I                                       S
         I                                      .
         I                                      .O
         I                                     ..
     400 +                                    ..
         I                                    .. U
         I                                   .. .
         I                                  .. .
         I                                  . .
         I         S .                     ...
     200 +             . .                ...
         I         *       . S            ..
         I           . .       .         ..
         I               . .      .     ..
         I                   * . .  .   .
         I                         . . *
       0 +--------------------------------------------------
                   10/17               10/19
                             10/18               12/12

                           DATE
```

Up to five variables can be plotted on the same vertical axis. When this is done, the scale on the vertical axis is determined based on the combined values of the vertical variables, and a separate point appears for each value of each variable.

When planning graphs with multiple variables or large numbers, adjust your variables so they are in the same order of magnitude. By redefining the variable plotted on the horizontal axis by a suitable power of 10 you can improve the readability of the finished graph. A method for doing this is shown in the example below.

```
DEFINE FILE SALES
SALES/D8.2=(UNIT_SOLD * RETAIL_PRICE)/10;
END

SET HISTOGRAM=OFF

GRAPH FILE SALES
HEADING CENTER
"STAMFORD'S SALES/10 AND RETURNS"
SUM SALES AND RETURNS ACROSS PROD_CODE
BY STORE
IF CITY IS 'STAMFORD'
FOOTING CENTER
"SALES FOR STORE # <STORE_CODE"
END
```



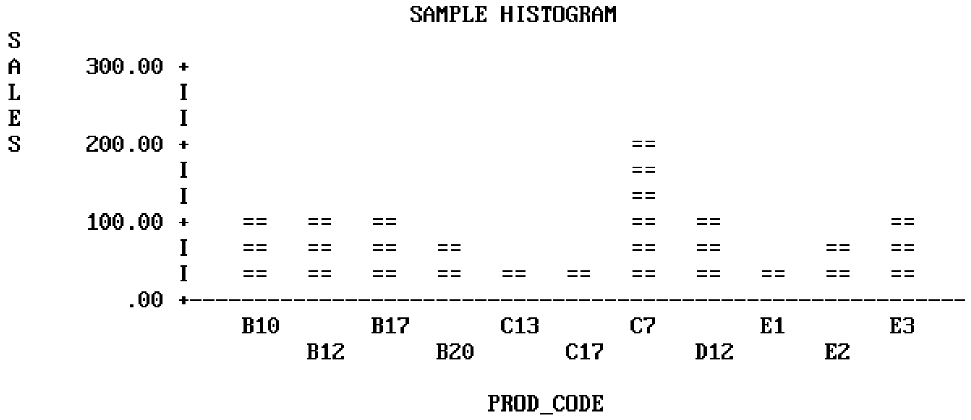
```
               STAMFORD'S SALES/10 AND RETURNS
    S  SALES
    R  RETURNS

15 +
   I
   I
   I
   I
   I                             S
10 +     R                      . .
   I      .                    .   .         R .
   I      .                   .     .       .S . . R
   I       .                  .      .     ..    . S
   I        .                .        .  . .
   I     S . .     . S .    .          S   .
 5 +         . S .       . S     R        .
   I          .              . .  .       .
   I           R           R       .     .
   I             . .   . .          .    .
   I                 R               .  .
   I                                  . .
 0 +-----------------------------------R------------------
         B10         B17         C7          E2
               B12         C13         D12         E3

                     PROD_CODE

                    SALES FOR STORE # 14B
```

Histograms

Histograms are vertical bar charts and are useful for portraying the component parts of aggregate values. They are essentially an alternate graphic format for plotting requests that could also generate connected point plots. To flip back and forth from one format to the other, simply reset the parameter HIST and issue REPLOT.

You create histograms by typing requests containing the verb SUM (or the synonyms, WRITE or ADD) and an ACROSS phrase that specifies an alphanumeric field. One bar appears on the graph for each verb object. The example that follows illustrates a histogram with a single variable.

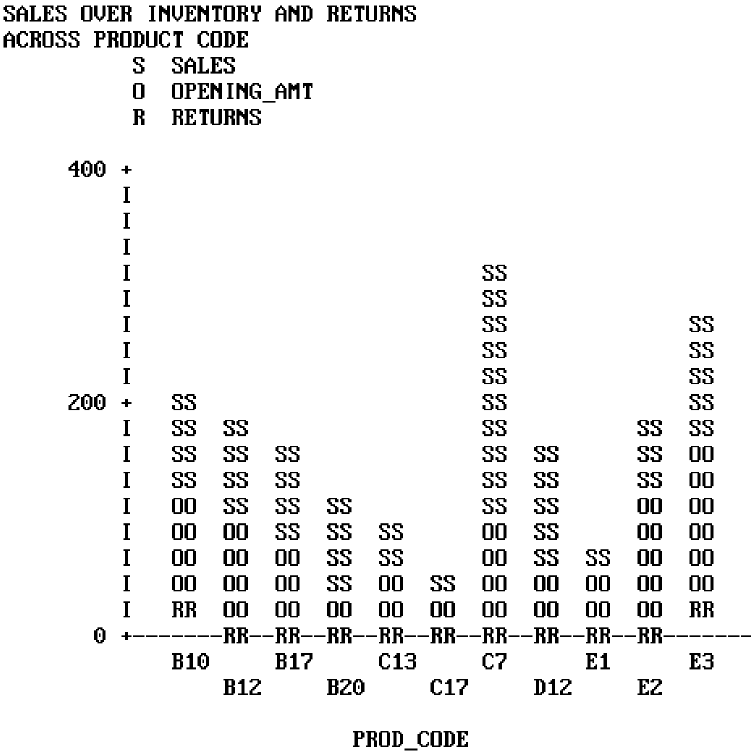
```
GRAPH FILE SALES
HEADING CENTER
"SAMPLE HISTOGRAM"
SUM SALES ACROSS PROD_CODE
END
```



```
                               SAMPLE HISTOGRAM
S
A     300.00 +
L            I
E            I
S     200.00 +                                  ==
             I                                  ==
             I                                  ==
      100.00 +    ==   ==   ==                  ==   ==             ==
             I    ==   ==   ==   ==             ==   ==        ==   ==
             I    ==   ==   ==   ==   ==   ==   ==   ==   ==   ==   ==
         .00 +-------------------------------------------------------------
                  B10       B17       C13       C7        E1        E3
                       B12       B20       C17       D12       E2

                                     PROD_CODE
```

To draw the bars side by side, separate the verb objects with spaces or AND. To draw superimposed (stacked) bars, separate the verb objects with OVER. The example that follows illustrates a request using OVER:

```
GRAPH FILE SALES
HEADING
"SALES OVER INVENTORY AND RETURNS"
"ACROSS PRODUCT CODE"
SUM SALES OVER INV OVER RETURNS ACROSS PROD_CODE
END
```



```
SALES OVER INVENTORY AND RETURNS
ACROSS PRODUCT CODE
          S  SALES
          O  OPENING_AMT
          R  RETURNS

     400 +
         I
         I
         I
         I                           SS
         I                           SS
         I                           SS              SS
         I                           SS              SS
         I                           SS              SS
     200 +   SS                      SS              SS
         I   SS  SS                  SS          SS  SS
         I   SS  SS  SS              SS  SS      SS  OO
         I   SS  SS  SS              SS  SS      SS  OO
         I   OO  SS  SS  SS          SS  SS      OO  OO
         I   OO  OO  SS  SS  SS      OO  SS      OO  OO
         I   OO  OO  OO  SS  SS      OO  SS  SS  OO  OO
         I   OO  OO  OO  SS  OO  SS  OO  OO  OO  OO  OO
         I   RR  OO  OO  OO  OO  OO  OO  OO  OO  OO  RR
       0 +-------RR--RR--RR--RR--RR--RR--RR--RR--RR-------
             B10     B17     C13     C7      E1      E3
                 B12     B20     C17     D12     E2

                           PROD_CODE
```

Note that the legend uses the full field names rather than the aliases for the verb objects (OPENING_AMT for INV).

When you name three or more verb objects in a request, you can have any combination of stacked and side-by-side bars.

Histogram Features

Each vertical bar or group of bars represents a value of the ACROSS sort field. The range of values for the verb objects determines the scale for the vertical axis.

All of the vertical axis features on histograms are adjustable:

- To reset the height of OFFLINE graphs, use the VAXIS parameter as described in *How to Set the Height* on page 19-42. (For online graphs, FOCUS automatically sets the height of your graph based on the terminal dimensions.)
- You can reset the upper and lower thresholds on the axis by setting the default scaling mechanism off (VAUTO) and setting new upper and lower limits (VMAX and VMIN). See *How to Set the Scale: Assigning Fixed Limits* on page 19-40.
- You can reset the class and tick intervals by overriding the default mechanism (AUTOTICK) and setting new intervals (VCLASS and VTICK). See *How to Set Class and Tick Intervals* on page 19-41.

FOCUS automatically sets the width of the bars and the spacing between them to fit within the HAXIS parameter limit. These can be changed by resetting the HAXIS parameter (see *How to Set the Width* on page 19-40).

The values for the data points on the HAXIS are printed horizontally on a single line or staggered (folded) on two or more lines, depending on the available space.

To add a grid of parallel horizontal lines at the vertical class marks, issue the following SET command before issuing your request:

```
SET GRID=ON
```

(Vertical grids are not available on histograms.)

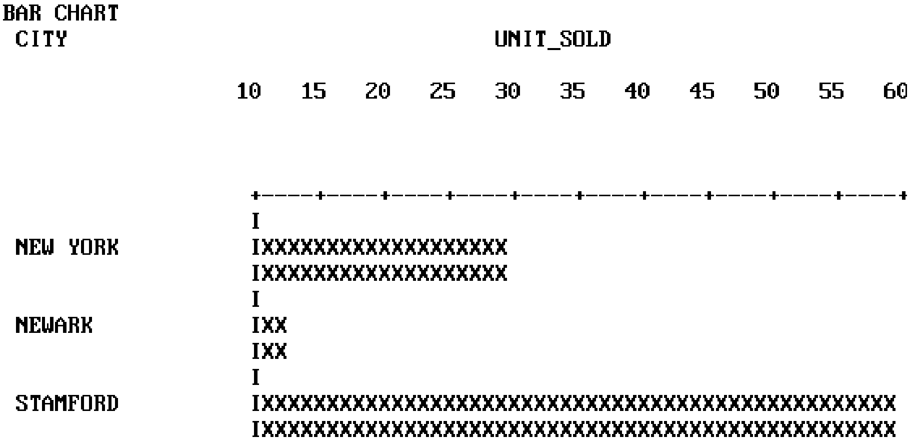
To specify stacking of all bars without using OVER in the request, you can set the parameter HSTACK (SET HSTACK=ON). (Remember to set it off again before moving to other requests.)

Note: There is often confusion over histogram features because of the similarity with bar charts. The BARNUMB facility used to print summary numbers for the bars in bar charts does not work with histograms.

Bar Charts

Bar charts have horizontal bars arrayed vertically. To produce a bar chart, type a request containing the verb SUM and a BY phrase (but no ACROSS phrase). A separate group of bars is created for each value of the BY field, and each group contains one bar for each verb object in the request.

```
SET BARWIDTH=2, BARSPACE=1

GRAPH FILE SALES
HEADING
"BAR CHART"
SUM UNIT_SOLD BY CITY
IF PROD_CODE EQ B10
END
```

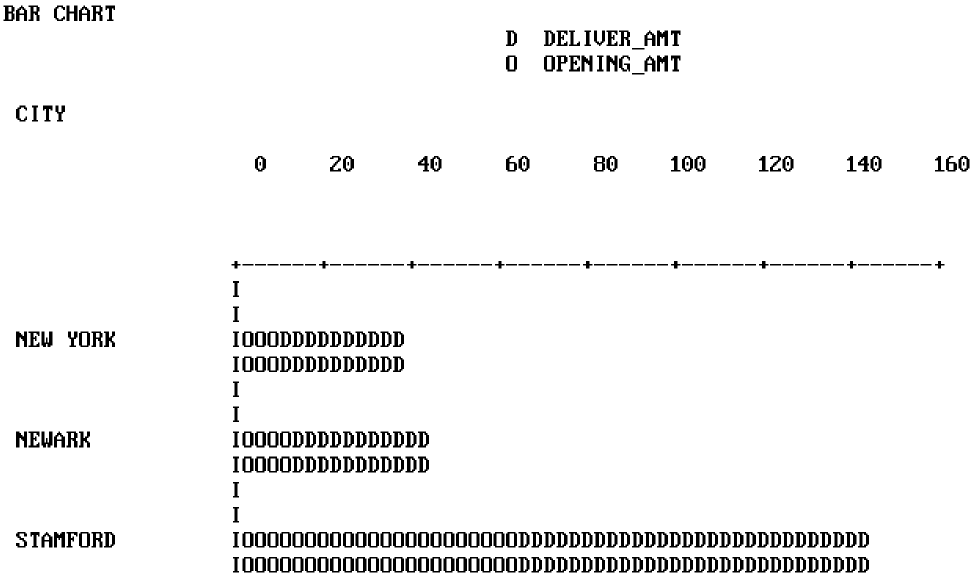


```
BAR CHART
 CITY                                 UNIT_SOLD

                  10   15   20   25   30   35   40   45   50   55   60



                   +----+----+----+----+----+----+----+----+----+----+
                   I
 NEW YORK          IXXXXXXXXXXXXXXXXXXX
                   IXXXXXXXXXXXXXXXXXXX
                   I
 NEWARK            IXX
                   IXX
                   I
 STAMFORD          IXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                   IXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

In the request above, the parameters BARSPACE and BARWIDTH were set to enhance the appearance of the graph and improve readability.

In requests with multiple verb objects, each bar appears beneath its predecessor by default. If verb objects are connected by OVER phrases, however, then the corresponding bars are stacked and appear end-to-end. The following example illustrates stacked bars.

```
SET BARSPACE=2, BARWIDTH=2

GRAPH FILE SALES
HEADING
"BAR CHART"
SUM DELIVER_AMT OVER INV BY CITY
WHERE 'PROD_CODE' EQ 'B10'
END
```



```
 BAR CHART
                                         D  DELIVER_AMT
                                         O  OPENING_AMT

  CITY

                     0     20     40     60     80    100    120    140    160



                   +------+------+------+------+------+------+------+------+
                   I
                   I
  NEW YORK         IOOODDDDDDDDDD
                   IOOODDDDDDDDDD
                   I
                   I
  NEWARK           IOOOODDDDDDDDDDD
                   IOOOODDDDDDDDDDD
                   I
                   I
  STAMFORD         IOOOOOOOOOOOOOOOOOOOOOODDDDDDDDDDDDDDDDDDDDDDDDDDDD
                   IOOOOOOOOOOOOOOOOOOOOOODDDDDDDDDDDDDDDDDDDDDDDDDDDD
```

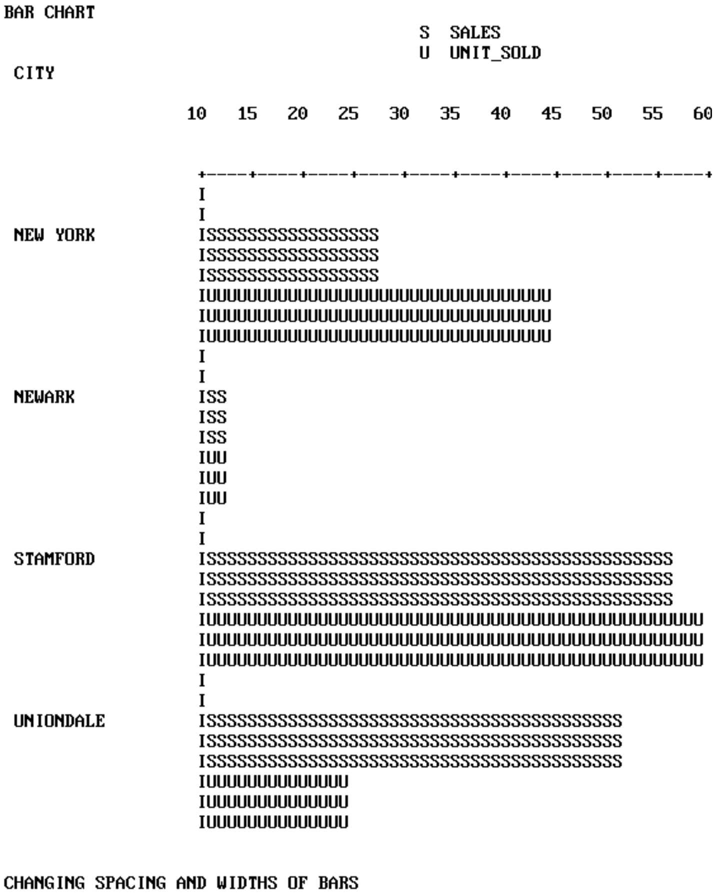
Alternatively, to request stacking of all bars, set the parameter BSTACK (SET BSTACK=ON). If you use BSTACK you do not need OVER; any graph can be replotted with and without stacking by simply changing the value of this parameter and issuing REPLOT.

Bar Chart Features

As we mentioned previously, you can set the BARWIDTH parameter to change the widths of the bars themselves and set the BARSPACE parameter to change the spacing between them. Set the GRID parameter to add a grid of vertical parallel lines at the class marks on the horizontal axis. The examples that follow illustrate the use of these parameters.

```
SET BARWIDTH=3, BARSPACE=2, BSTACK=OFF

GRAPH FILE SALES
HEADING
"BAR CHART"
SUM AVE.SALES AND UNIT_SOLD BY CITY
WHERE 'PROD_CODE' IS 'B10' OR 'B20'
FOOTING
"</2 CHANGING SPACING AND WIDTHS OF BARS"
END
```

The result appears below.

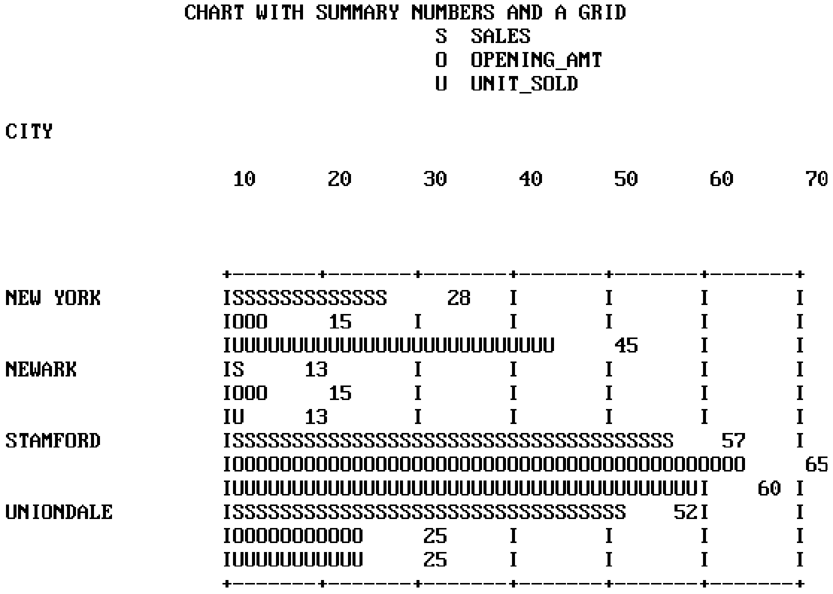


```
BAR CHART
                                        S  SALES
                                        U  UNIT_SOLD
 CITY

                  10   15   20   25   30   35   40   45   50   55   60


                   +----+----+----+----+----+----+----+----+----+----+
                   I
                   I
 NEW YORK          ISSSSSSSSSSSSSSSSS
                   ISSSSSSSSSSSSSSSSS
                   ISSSSSSSSSSSSSSSSS
                   IUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                   IUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                   IUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                   I
                   I
 NEWARK            ISS
                   ISS
                   ISS
                   IUU
                   IUU
                   IUU
                   I
                   I
 STAMFORD          ISSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
                   ISSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
                   ISSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
                   IUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                   IUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                   IUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                   I
                   I
 UNIONDALE         ISSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
                   ISSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
                   ISSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
                   IUUUUUUUUUUUUUU
                   IUUUUUUUUUUUUUU
                   IUUUUUUUUUUUUUU


CHANGING SPACING AND WIDTHS OF BARS
```

To print a summary value at the end of each bar, set the BARNUMB parameter. **Note:** This feature is also available on pie charts, but is not available on histograms.

The effects of BARNUMB and GRID are shown below.

```
SET BARNUMB=ON, GRID=ON
GRAPH FILE SALES
HEADING CENTER
"CHART WITH SUMMARY NUMBERS AND A GRID"
SUM AVE.SALES AND INV AND UNIT_SOLD BY CITY
WHERE 'PROD_CODE' EQ 'B10' OR 'B20'
END
```



```
               CHART WITH SUMMARY NUMBERS AND A GRID
                                    S  SALES
                                    O  OPENING_AMT
                                    U  UNIT_SOLD

CITY

                   10      20      30      40      50      60      70



                  +-------+-------+-------+-------+-------+-------+-------+
NEW YORK          ISSSSSSSSSSSSS     28   I       I       I       I
                  IOOO     15     I       I       I       I       I
                  IUUUUUUUUUUUUUUUUUUUUUUUUUUU     45     I       I
NEWARK            IS     13       I       I       I       I       I
                  IOOO     15     I       I       I       I       I
                  IU     13       I       I       I       I       I
STAMFORD          ISSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    57    I
                  IOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO     65
                  IUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUI    60 I
UNIONDALE         ISSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    52I       I
                  IOOOOOOOOOOO     25     I       I       I       I
                  IUUUUUUUUUUU     25     I       I       I       I
                  +-------+-------+-------+-------+-------+-------+-------+
```

The horizontal axis features are all adjustable:

- To change the width of OFFLINE graphs, alter the HAXIS parameter as described in *How to Set the Width* on page 19-40. (For ONLINE graphs, FOCUS will automatically detect the width of the terminal and display the graph accordingly.)
- To reset the numerical scale, turn off the default scaling mechanism (HAUTO) and set new upper and lower limits (HMAX and HMIN). See *How to Set the Scale: Assigning Fixed Limits* on page 19-42.
- To change the class and tick intervals, override the default mechanism (AUTOTICK) and set new intervals (HCLASS and HTICK). See *How to Set Class and Tick Intervals* on page 19-41.

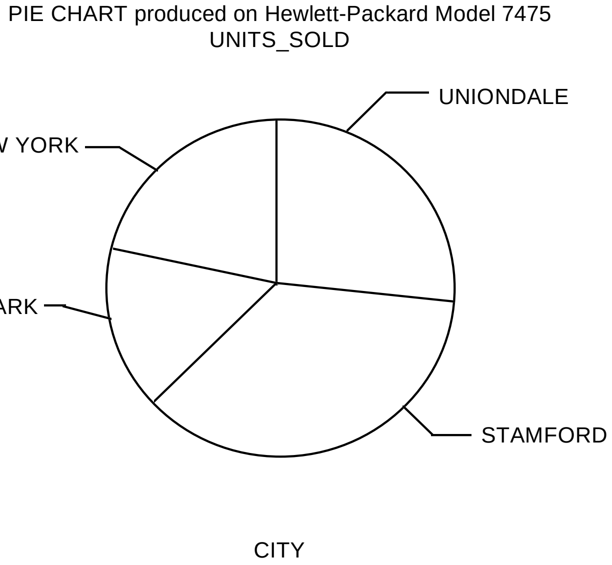
The vertical axis length is controlled by FOCUS. You can set the bar widths and spacing as mentioned previously, but you cannot set the vertical height to a fixed dimension.

Pie Charts

Pie charts can only be drawn on high-resolution graphic devices. (It is possible, however, to create a formatted pie chart and save it for subsequent plotting on another device. See *Saving Formatted GRAPH Output* on page 19-50.)

To create a pie chart, first set the PIE parameter ON and select a device (SET DEVICE=), then type a request with the verb SUM (or the synonyms, WRITE or ADD) and an ACROSS phrase that names an alphanumeric field. When you finish your pie charts, set the PIE parameter OFF before running other types of GRAPH requests.

```
SET PIE=ON, DEVICE=HP7220C

GRAPH FILE SALES
HEADING CENTER
"PIE CHART PRODUCED ON HEWLETT-PACKARD MODEL 7475"
WRITE RPCT.UNIT_SOLD ACROSS CITY
END
```

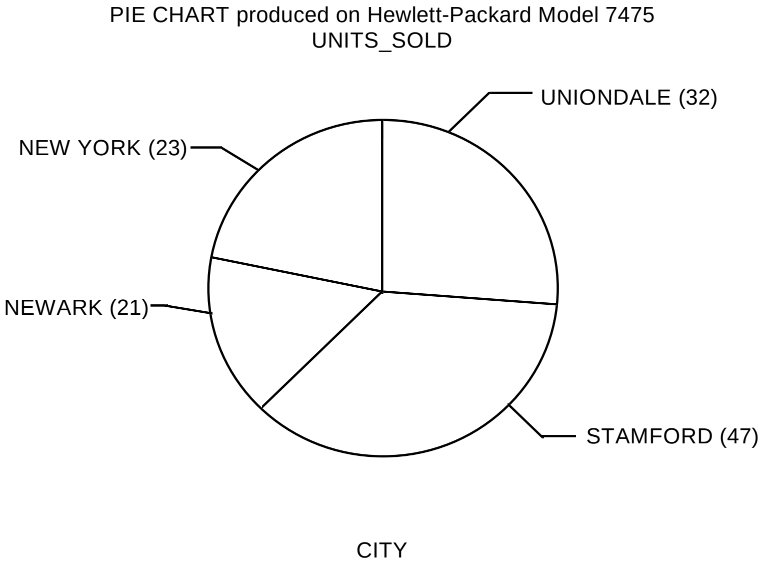


Pie Chart Features

To add summary numbers for each slice of the pie chart on the previous page, simply enter the following commands:

```
SET BARNUMB=ON
REPLOT
```

The effect is shown below:

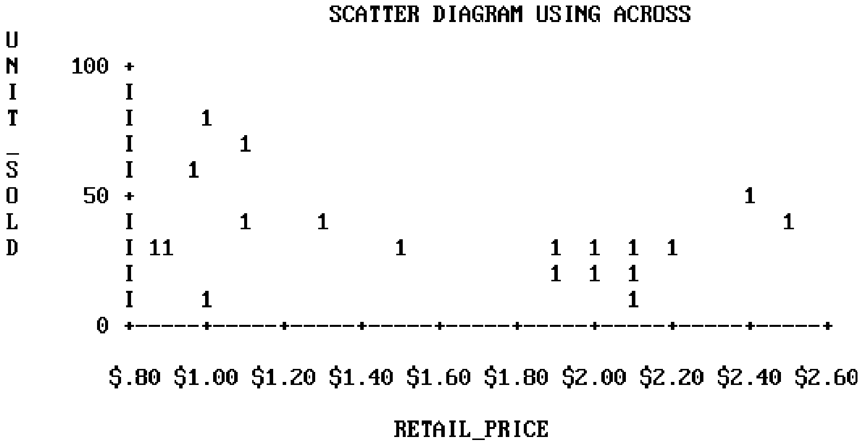


Note: FOCUS does not include a facility for displaying exploded pie chart slices.

Scatter Diagrams

Scatter diagrams illustrate occurrence patterns and distribution of variables. You create them by issuing requests containing the verb PRINT and a sort phrase (BY or ACROSS). The choice of BY or ACROSS dictates the vertical or horizontal bias of the graph. The samples that follow illustrate both types.

```
GRAPH FILE SALES
HEADING CENTER
"SCATTER DIAGRAM USING ACROSS"
PRINT UNIT_SOLD ACROSS RETAIL_PRICE
END
```

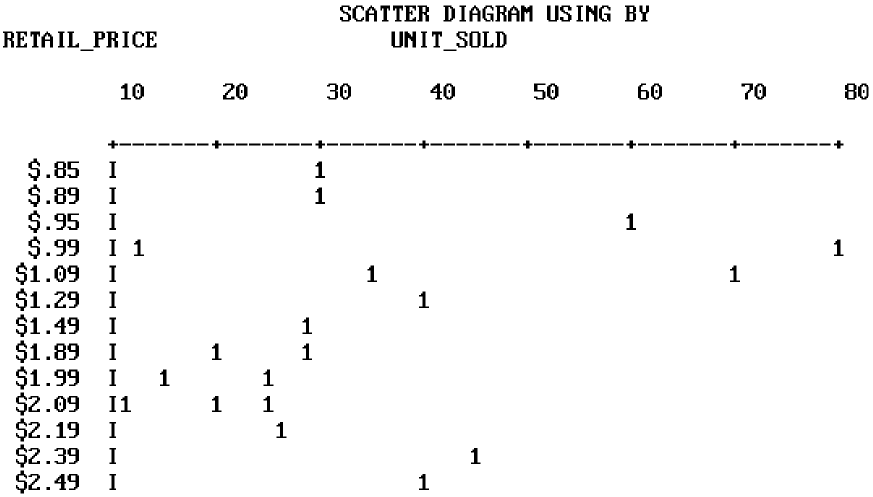


```
                         SCATTER DIAGRAM USING ACROSS
U
N    100 +
I        I
T        I     1
_        I        1
S        I    1
O     50 +                                               1
L        I        1     1                                   1
D        I 11                 1           1  1  1  1
         I                                1  1  1
         I     1                                1
       0 +-----+-----+-----+-----+-----+-----+-----+-----+-----+

        $.80 $1.00 $1.20 $1.40 $1.60 $1.80 $2.00 $2.20 $2.40 $2.60

                              RETAIL_PRICE
```

The point plots on the vertical axis represent the values for the ACROSS field named. Each record selected contributes a separate point. The sort control fields are plotted on the horizontal axis, which is also scaled if the control field values are numeric.

When the request contains a BY phrase, the named sort control field is plotted down the vertical axis and the data values are scaled horizontally.

```
GRAPH FILE SALES
HEADING CENTER
"SCATTER DIAGRAM USING BY"
PRINT UNIT_SOLD BY RETAIL_PRICE
END
```



```
                          SCATTER DIAGRAM USING BY
RETAIL_PRICE                  UNIT_SOLD

         10      20      30      40      50      60      70      80

        +-------+-------+-------+-------+-------+-------+-------+
  $.85  I               1
  $.89  I               1
  $.95  I                                       1
  $.99  I 1                                                     1
 $1.09  I                   1                           1
 $1.29  I                       1
 $1.49  I              1
 $1.89  I       1      1
 $1.99  I   1       1
 $2.09  I1      1   1
 $2.19  I            1
 $2.39  I                           1
 $2.49  I                       1
```

The vertical axis is not scaled even if the control field is numeric. Each separate value of the control field is plotted on a different line, but these are not arranged according to a numerical scale. The full range of horizontal scaling options is available (see *The Horizontal Axis: System Defaults* on page 19-39).

Scatter Diagram Features

When multiple points fall in the same position, FOCUS displays either a number (for up to nine occurrences) or an asterisk (for more than nine occurrences).

When you specify more than one verb object (five are permitted), they are represented by the first letter of the field name. If they are not different you can assign unique symbols with AS phrases.

Scatter diagrams can display the following:

- Trend lines (available only in plots generated using ACROSS). Trend lines are calculated by Ordinary Least Squares (OLS) regression analysis and represent the line of best fit. You can add them to requests containing ACROSS phrases by setting the parameter GTREND before executing or replotting the request:

  ```
  SET GTREND=ON
  ```

  When two fields are plotted with GTREND=ON, FOCUS provides two trend lines. If more than two fields are plotted, however, FOCUS does not provide trend lines.

- Horizontal grids. You can add horizontal grid lines at the vertical class marks by setting the parameter GRID:

  ```
  SET GRID=ON
  ```

- Vertical grids (available only in plots generated by requests using BY). You can add vertical parallel lines at the horizontal class marks of the scatter plot by setting the parameter VGRID:

  ```
  SET VGRID=ON
  ```

Adjusting Graph Elements

All graphs other than pie charts have horizontal and vertical axes. These axes usually have scales with adjustable upper and lower thresholds that are divided into class intervals representing quantities of data (scales are only provided when the variables named are computational fields). Class intervals are further broken down with tick marks representing smaller increments of data.

When multiple graphs are created in a single request, FOCUS determines the default horizontal scale after examining all values to be plotted, and the same scale is then applied to each graph. Vertical scales are recalculated each time, however, and adjusted for the values in each graph (unless you override this feature).

Some graph forms, notably connected point plots, histograms, and bar charts, can be visually strengthened by the addition of parallel lines across the horizontal and/or vertical axes, to form a grid against which the data is arrayed.

The material that follows describes the default conditions for all of these graph features, and the facilities for changing the default values to create customized output.

At any time during your session you can review the current GRAPH parameter settings by typing

```
? SET GRAPH
```

which displays the current settings of all of the adjustable GRAPH parameters, as shown below.

Parameter	**Setting**
`DEVICE`	IBM3270
`GPROMPT`	OFF
`GRID`	OFF
`VGRID`	OFF
`HAXIS`	130
`VAXIS`	66
`GTREND`	OFF
`GRIBBON (GCOLOR)`	OFF
`VZERO`	OFF
`VAUTO`	ON
`VMAX`	.00
`VMIN`	.00
`HAUTO`	ON
`HMAX`	.00
`HMIN`	.00

Parameter	Setting
AUTOTICK	ON
HTICK	.00
HCLASS	.00
VTICK	.00
VCLASS	.00
BARWIDTH	1
BARSPACE	0
BARNUMB	OFF
HISTOGRAM	ON
HSTACK	OFF
BSTACK	OFF
PIE	OFF
GMISSING	OFF
GMISSVAL	.00

Figure 19-10. GRAPH Parameter Settings

For information about each of the parameters listed, refer to *SET Parameters* on page 19-59.

The Horizontal Axis: System Defaults

The width of each graph, including any surrounding text, is controlled by the HAXIS parameter. For online displays, FOCUS automatically detects the terminal width and plots the graph accordingly.

For graphs generated OFFLINE, the default value for HAXIS is normally set to the maximum possible size for the output device selected, after allowing for the inclusion of any text required for the vertical axis and its labels along the left margin. To maximize display space, you can limit the size of your labels through the use of either AS phrases or DECODE expressions.

In setting the scale (when AUTOTICK=ON, and HAUTO=ON), FOCUS determines the amount of available space and the range of values selected for plotting. It then selects minimum, intermediate, and maximum unit values for the horizontal axis scale that encompass the range of values and are convenient multiples of an appropriate power of 10 (10 vs. 1000 vs. 1,000,000).

When you select a high-resolution graphic device, FOCUS controls the axis dimensions according to the values shown for the various devices in *SET Parameters* on page 19-59.

Syntax How to Set the Width

To set the width of the graph to a given number of characters, issue the SET statement

```
SET HAXIS=nn
```

where *nn* is a numeric value between 20 and 130.

Syntax How to Set the Scale: Assigning Fixed Limits

FOCUS automatically sets the horizontal scale to cover the total range of values to be plotted (HAUTO=ON). The range is divided into intervals called classes. The scale is normalized to provide class values rounded to the appropriate multiples and powers of 10 for the intervals plotted on the axis.

If you wish to assign fixed upper and lower limits (useful when producing a series of graphs where consistent scales are needed), you do so by turning off the automatic scaling mechanism and setting new limit values. This is done with the SET command. The syntax is as follows

```
SET HAUTO=OFF, HMAX=nn, HMIN=nn
```

where:

`HAUTO`
: Is the automatic scaling facility.

`HMAX`
: Is the parameter for setting the upper limit on the horizontal axis. The default is 0.

`HMIN`
: Is the parameter that controls the lower limit on the horizontal axis when HAUTO is OFF. The default is 0.

`nn`
: Is the new limit.

Note:

- When entering several SET parameters on one line, separate them with commas.
- If you define limits that do not incorporate all of the data values, FOCUS displays OVER and/or UNDER to indicate that some of the data extracted is not reflected on the graph.

Syntax How to Set Class and Tick Intervals

Class intervals are the intervals between the labels and grid lines on a graph. Tick intervals are the subdivisions of class intervals. When AUTOTICK is ON, FOCUS automatically determines the class and tick intervals.

To set the class and tick intervals yourself, first turn off the default scaling mechanism; then reset the class and tick intervals with the SET command

```
SET AUTOTICK=OFF, HCLASS=nn, HTICK=nn
```

where:

`AUTOTICK`

Is the automatic scaling mechanism.

`HCLASS`

Is the parameter that controls the class interval on the horizontal axis when AUTOTICK is OFF. The default is 0.

`nn`

Is the new class interval value for the axis.

`HTICK`

Is the parameter that controls the tick interval when AUTOTICK is OFF. The default is 0.

`nn`

Is the new tick interval for the axis.

Note:

- When issuing more than one parameter with a sample SET command, separate parameters with commas as shown above.
- To make the changes apparent on the screen, SET SCREEN to PAPER.
- The number of ticks per class is HCLASS/HTICK.

The Vertical Axis: System Defaults

The vertical axis (VAXIS) represents the number of lines in the graph, including any surrounding text.

For online displays, FOCUS automatically plots the graph according to the terminal height. For graphs generated offline, FOCUS respects VAXIS settings.

FOCUS automatically sets the vertical scale to cover the total range of values to be plotted (VAUTO=ON). The height is set as high as possible (taking into consideration the presence of any headings and/or footings, and the need to provide suitably rounded vertical class markers).

The range is divided into intervals called classes. The scale is normalized to provide class values rounded to the appropriate multiples and powers of 10 for the intervals plotted on the axis.

As with the horizontal axis, FOCUS selects the vertical axis size whenever you select a high-resolution graphic device (see *SET Parameters* on page 19-59).

Syntax How to Set the Height

Use the following SET command to set the vertical axis

```
SET VAXIS=nn
```

where *nn* is a number in the range 20-66.

Syntax How to Set the Scale: Assigning Fixed Limits

If you wish to give the vertical scale fixed upper and lower limits (useful when producing a series of graphs where consistent scales are needed), you can do so by turning off the automatic scaling mechanism and setting fixed limits. This is done with the SET command

```
SET VAUTO=OFF, VMAX=nn, VMIN=nn
```

where:

`VAUTO`

Is the automatic scaling facility.

`VMAX`

Is the parameter for setting the upper limit on the vertical axis. The default is 0.

`VMIN`

Is the parameter that controls the lower limit on the vertical axis when VAUTO is OFF. The default is 0.

nn

Is the new limit.

Note:

- When entering several SET parameters on one line, separate them with commas.
- If you define limits that do not incorporate all of the data values, FOCUS displays OVER and/or UNDER to indicate that some of the data extracted is not reflected on the graph.

Syntax How to Set Class and Tick Intervals

To set the class and tick intervals on the vertical axis, first turn off the default scaling mechanism, and then reset the class and tick intervals with the SET command

```
SET AUTOTICK=OFF, VCLASS=nn, VTICK=nn
```

where:

`AUTOTICK`
: Is the automatic scaling mechanism.

`VCLASS`
: Is the parameter that controls the class interval on the vertical axis when AUTOTICK is OFF. The default is 0.

`nn`
: Is the new class interval for the vertical axis.

`VTICK`
: Is the parameter that controls the tick interval when AUTOTICK is OFF. The default is 0.

`nn`
: Is the new tick interval for the axis.

Note:

- When setting more than one parameter, separate them with commas.
- To make the changes apparent on screen, SET SCREEN to PAPER.
- The number of ticks per class is VCLASS/VTICK.

Highlighting Facilities

FOCUS contains several facilities for highlighting the information shown on your graphs. Specifically, these include the following:

- Grid lines can be added on one or both axes of connected point plots and scatter diagrams or the horizontal axis of histograms.
- Trend lines are usually included on most scatter plots.
- Summary numbers can be printed for each slice of a pie chart or bar on a bar chart.

Syntax How to Add Horizontal or Vertical Grids

Grids often make graphs easier to read. They are parallel lines drawn across the graph at the vertical and/or horizontal class marks on the axes.

Horizontal grid lines are available on connected point plots, histograms, and scatter diagrams. To add them at the vertical class marks on your graph, issue the following command:

```
SET GRID=ON
```

Vertical grid lines are available only on high-resolution devices in requests for connected point plots and scatter diagrams, and only when the values on both axes are numeric. To add them at the horizontal class marks on the graph, issue the following command:

```
SET VGRID=ON
```

To remove the lines, set the appropriate parameter OFF.

Syntax How to Add Summary Numbers in Pie and Bar Charts

To print a summary number at the end of each bar on a bar chart or in each slice of a pie chart, set the parameter BARNUMB:

```
SET BARNUMB=ON
```

These summary numbers are not available on histograms.

Syntax How to Add Trend Lines on Scatter Plots

Trend lines are useful on scatter plots to give a focus to the sometimes confusing array of plot points. The trend line represents the notion of the “best fit” calculated by Ordinary Least Squares (OLS) regression analysis.

When two data fields are scattered across the same horizontal axis, each is given its own trend line. On some terminals with two-color ribbons the lines are differentiated by color.

The system always requests a value for the parameter GTREND, whenever a scatter diagram is requested (the default value for GTREND is OFF). To request a trend line set GTREND on:

```
SET GTREND=ON
```

Special Topics

There are a few topics that have general applicability for many graph applications. Specifically, they are:

- How does FOCUS handle dates in graphs?
- How is missing data handled?
- Is it possible to save formatted graphic output and display it later?
- Is it possible to send graphs to a Personal Computer for display?
- What is the nature of the interface between FOCUS and CA-TELLAGRAF?
- What is the nature of the interface between FOCUS and ICU (Interactive Chart Utility)?

These are described in the following sections.

Plotting Dates

Numerical fields containing dates are recognized by FOCUS through the formats in their Master Files. Such fields are interpreted by FOCUS if you name them in ACROSS or BY phrases in GRAPH requests. To review the various format types, see the *Describing Data* manual.

When plotting dates, FOCUS handles them in the following manner:

- If the date field named has a month format, it is plotted in ascending time order (even though the file is not sorted in ascending date order). Hence, month/year values of 01/76, 03/76, 09/75 will be plotted by month within year: 09/75, 01/76, 03/76.
- Axis scaling is performed on the basis of days in the month and months in the year. When the date format includes the day, the scale usually starts at the first day of the month as the zero axis point.

In some instances you may wish to selectively combine groups of date point plots to reduce the number of separate points on the horizontal axis. You do this with the IN-GROUPS-OF option. For example, if the date field format is I6YMD you can display the data by month rather than by day by grouping it in 30-day increments:

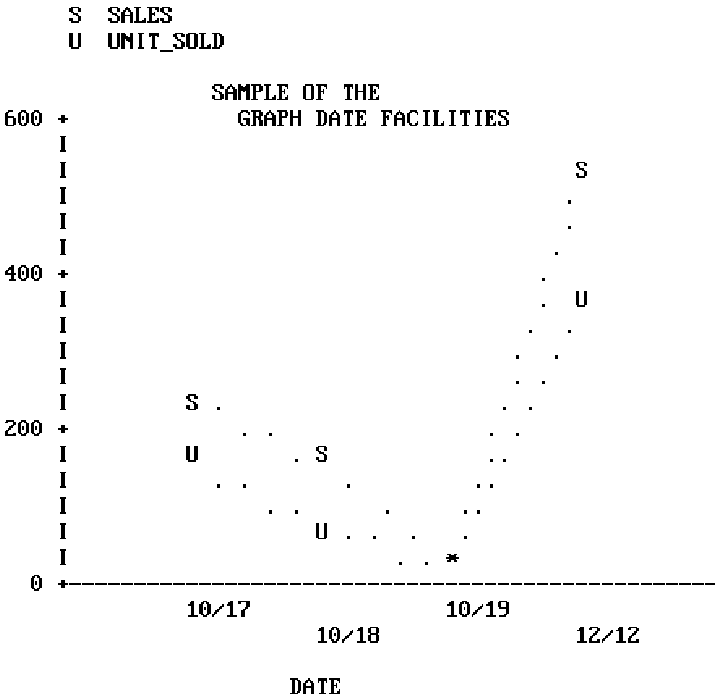
```
ACROSS DATE IN-GROUPS-OF 30
```

This eliminates plot points for individual days. If your date format is in a legacy YMD format you could also redefine the format and divide the field contents by 100 to eliminate the days:

```
DATE/I4YM=DATE/100
```

The example that follows illustrates a graph with date plots.

```
GRAPH FILE SALES
HEADING
"</6 <22 SAMPLE OF THE"
"<24 GRAPH DATE FACILITIES"
SUM SALES AND UNIT_SOLD ACROSS DATE
END
```



```
     S  SALES
     U  UNIT_SOLD

                SAMPLE OF THE
600 +             GRAPH DATE FACILITIES
    I
    I                                       S
    I                                      .
    I                                      .
    I                                     .
400 +                                    .
    I                                    .  U
    I                                   .  .
    I                                  .  .
    I                                  . .
    I         S .                     . .
200 +             . .                . .
    I         U       . S            ..
    I           . .       .         ..
    I               . .      .     ..
    I                   U . .  .   .
    I                         . . *
  0 +---------------------------------------------------
              10/17               10/19
                        10/18               12/12

                      DATE
```

Handling Missing Data

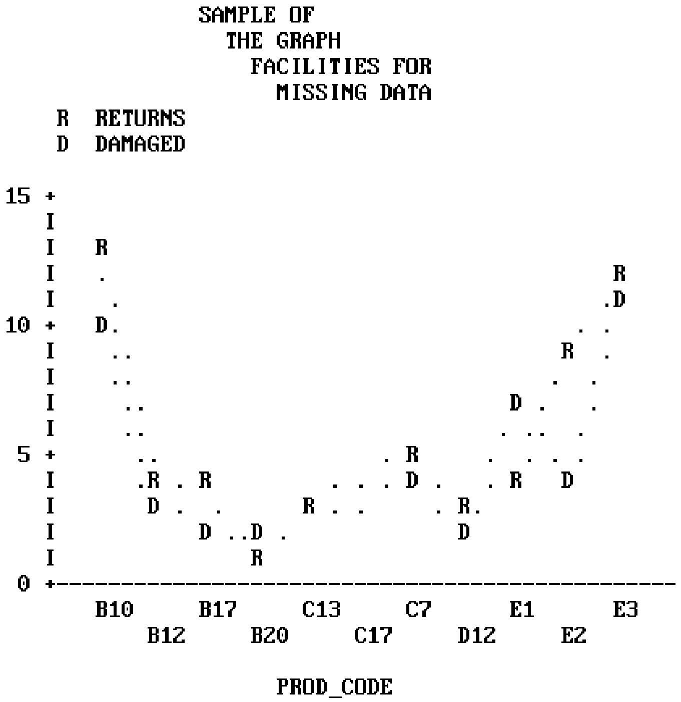
You can handle missing data selectively in GRAPH requests. You can portray the missing data as null values or you can choose to ignore missing values and have the plot span the missing points. This applies to requests containing both ACROSS and BY phrases, where the ACROSS values are plotted across the horizontal axis.

Normally, missing values on the vertical axis are ignored (VZERO=OFF). If ON, the values are treated as zero (0).

You instruct the system to ignore missing values through the SET options, GMISSING and GMISSVAL, or you can set GPROMPT=ON, and select the processing of the missing values when you execute your request. These SET operations can be done once for your entire session, or may be done on an individual basis to refine a particular request. Keep in mind that they remain in effect until you reset the parameters (see *SET Parameters* on page 19-59).

The examples that follow illustrate the same request, but with different treatments of missing values selected.

```
SET GMISSING=ON, HIST=OFF

GRAPH FILE SALES
HEADING
"</1 <22 SAMPLE OF"
"<24 THE GRAPH "
"<26 FACILITIES FOR"
"<28 MISSING DATA"
SUM RETURNS AND DAMAGED ACROSS PROD_CODE
END
```

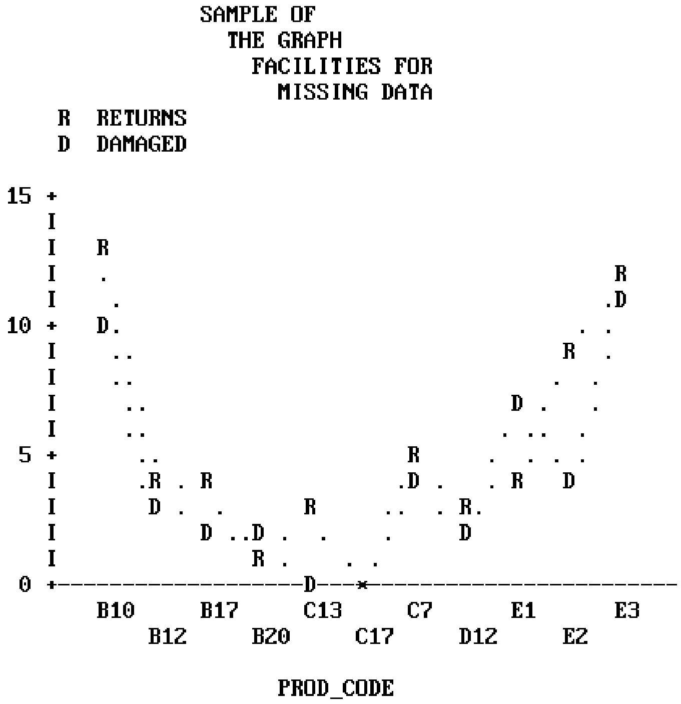


```
               SAMPLE OF
                 THE GRAPH
                   FACILITIES FOR
                     MISSING DATA
    R  RETURNS
    D  DAMAGED

15 +
   I
   I   R
   I   .                                       R
   I    .                                     .D
10 +   D.                                   . .
   I    ..                                 R  .
   I    ..                                .  .
   I     ..                            D .   .
   I     ..                           . ..  .
 5 +      ..                 . R     .  . . .
   I      .R . R         . . . D .   . R   D
   I       D .  .      R . .     . R.
   I           D ..D .             D
   I               R
 0 +------------------------------------------------
       B10     B17     C13     C7      E1      E3
           B12     B20     C17     D12     E2

                     PROD_CODE
```

In this example GMISSING is ON and GMISSVAL is 0, so the graph ignores zero values for products C13 and C17.

The graph below shows the effect of changing the GMISSING parameter to OFF.

```
SET GMISSING=OFF

REPLOT
```



```
               SAMPLE OF
                 THE GRAPH
                   FACILITIES FOR
                     MISSING DATA
    R  RETURNS
    D  DAMAGED

15 +
   I
   I   R
   I   .                                       R
   I    .                                     .D
10 +   D.                                   . .
   I    ..                                 R  .
   I    ..                                .  .
   I     ..                            D .   .
   I     ..                           . ..  .
 5 +      ..                   R     .  . . .
   I      .R . R              .D .   . R   D
   I       D .  .      R     ..  . R.
   I           D ..D .  .    .     D
   I               R .    . .
 0 +-------------------D---*-------------------------
       B10     B17     C13     C7      E1      E3
           B12     B20     C17     D12     E2

                     PROD_CODE
```

The values for products C13 and C17 were shown as positive values with GMISSING ON. With GMISSING=OFF, the zero values for products C13 and C17 are plotted on the graph.

Using Fixed-Axis Scales

When creating series of graphs it is often desirable to have the same horizontal and vertical scales used for each graph in the group. This situation arises whenever your graph request combines an ACROSS phrase with a BY phrase.

In such requests the ACROSS values are plotted across the horizontal axis and a separate graph is created for each value of the BY field. The default scales for the graphs will vary depending on the range of values for each verb object and BY field combination.

To apply the same scale to each graph, turn off the default scaling mechanisms, and define your own minimum and maximum values for the axis thresholds (see *How to Set the Scale: Assigning Fixed Limits* on page 19-40 in the section *The Horizontal Axis: System Defaults,* and *How to Set the Scale: Assigning Fixed Limits* on page 19-42 in the section *The Vertical Axis: System Defaults*).

Saving Formatted GRAPH Output

You can place the output from GRAPH commands into specially formatted SAVE files for subsequent conversion into printed or displayed graphs. This capability (called deferred output) is useful for developing graph requests on a device other than the one you will use to produce the final graph.

The facility described below is available for all ASCII graphics devices that FOCUS supports, but is not available for the IBM 3279 color graphics terminal, which has a separate GDDM facility for this purpose (see *IBM Devices Using GDDM* on page 19-54). In addition, deferred output cannot be generated from a CONSOLE.

The syntax for the FOCUS facility is

```
GRAPH FILE ...
SUM ...
.
.
.
ON  {GRAPH|TABLE}   SAVE [AS savename] FORMAT GRAPH
ON  {GRAPH|TABLE}   SET parameter value [, parameter value...]
END
```

where:

`ON GRAPH|ON TABLE`

Denotes the command environment from which the request is entered. This syntax suppresses display of the output and returns a message that the file has been saved.

`SAVE|SET`

Is the action taken.

`AS` *`savename`*

Is an optional parameter that allows you to assign a permanent file name as the target for the formatted output. The default is FOCSAVE.

`parameter value`

Is the system value you want to change or set. Any parameter discussed in the *Developing Applications* manual can be set or changed here. The syntax is essentially the same as ON TABLE SET, which is discussed in Chapter 4, *Sorting Tabular Reports*.

`FORMAT GRAPH`

Specifies that the output is to be formatted for whatever graphics device is specified in the DEVICE parameter (see *SET Parameters* on page 19-59), and saved in either the SAVE file or a file you name in an AS phrase.

As an alternative, you can display a graph on the CONSOLE before creating a specially formatted SAVE file. To use this facility, enter a GRAPH request to generate a display, as shown below:

```
GRAPH FILE ...
SUM ...
    .
    .
    .
END
```

After viewing the graph, use the following syntax to save the graph for later output on another device:

```
SAVE [AS savename] FORMAT GRAPH
```

Syntax How to Display Stored Graphs

To display stored graphs, issue the appropriate form of the REPLOT command from the output graphics DEVICE

```
REPLOT  [GRAPH|FROM]   ddname
```

where:

`REPLOT [GRAPH|FROM]`
Is the function to be performed.

`ddname`
Is the SAVE file name. This must be provided even if the default FOCSAVE file was used.

Note:

- You need not redefine the graphics device with another SET command. The device specified through the DEVICE= parameter when the graph was saved still applies.
- You can save the internal matrix produced for a request and issue a REPLOT later in the session if SAVEMATRIX is set to ON (see the *Developing Applications* manual).

You can allocate the file yourself through the appropriate operating system procedure or you can let FOCUS allocate the SAVE file for you dynamically. If you allow FOCUS to allocate the file it will allocate a temporary file that you must rename if you wish to keep it after you log off.

The record layout of the graphics SAVE file is documented in Technical Memorandum #7704 *Description of Deferred Graphics Output* (available through your Information Builders Branch Office). You can process this file yourself if you have a deferred graph system that accepts low-level terminal graphics commands.

Displaying Graphs With PC/FOCUS or FOCUS for Windows

If you have FOCUS on the mainframe and PC/FOCUS® on your personal computer and an appropriate communications link, you can extract data from the mainframe files and send the extract file to the personal computer where you actually issue the GRAPH request in PC/FOCUS or FOCUS for Windows. This is a useful option because the printers attached to personal computers often create more attractive graphs than those produced on the system high-speed printer. FOCUS file transfer facilities are described in the *Developing Applications* manual.

Note, however, that you cannot send formatted graphs to the personal computer for plotting.

Creating Formatted Input for CA-TELLAGRAF

The Interface to CA-TELLAGRAF is a separate optional interface product that you use to create formatted FOCUS output files ready for processing by CA-TELLAGRAF, the publication-quality graphics system produced by Computer Associates.

With it, you can write FOCUS GRAPH requests that generate files containing actual CA-TELLAGRAF commands and all of the necessary data and control information for producing graphs.

The data may originate in any FOCUS file or any file that FOCUS can read (for example, QSAM, VSAM, ISAM, IMS, CA-IDMS/DB, ADABAS, TOTAL, SQL, SYSTEM 2000, Model 204).

Directions for using the Interface can be found in the *TELLAGRAF Interface Users Manual.*

Using the FOCUS ICU Interface

The FOCUS ICU Interface is a separate optional interface product that you can use to generate graphs in conjunction with IBM’s Interactive Chart Utility.

ICU displays graphs and provides menu selections which allow you to change such factors as graph type, size, and legend, and to send the graph to a printer.

The ICU Interface can place you directly in the ICU environment or can save the graph format and data for subsequent ICU processing. When you leave ICU, control is returned to FOCUS.

All ICU graphics requests follow the standard FOCUS rules and each of the default graphs is represented by an ICU format file distributed with FOCUS.

To use the ICU Interface, issue the command:

```
SET DEVICE = ICU
```

Subsequent GRAPH requests will use ICU to generate graphs.

Directions for using this Interface can be found in the *ICU Interface Users Manual.*

Special Graphics Devices

Graphs created with the FOCUS graphics generator can be printed or displayed in three levels of detail:

- Low-resolution graphs are produced by high-speed line printers and non-graphics terminals. Normally, this is adequate graphic information. While such graphs are not elegant, they are easily produced and allow you to preview graphic scenarios and refine the shapes and contents of your graphs. Subsequently, to create more “finished” versions you need only choose a different device or save the formatted output in a file to print later when a high-resolution device is available.
- Medium-resolution graphs are produced on devices such as Diablo, Trendata, and Anderson-Jacobson printers. These devices, which are driven by step motors, draw nearly continuous line plots, but the quality is not adequate for presentations.
- High-resolution graphic devices print continuous line plots, smooth curves, and create presentation-quality graphs. This category includes both devices created specifically for displaying graphics images (flat-bed and continuous line plotters, and color printers), as well as color CRTs. FOCUS supports three types of high-resolution graphics devices:
  - Hewlett-Packard four- and eight-pen plotters.
  - IBM graphic CRTs and printers.
  - Tektronix CRTs.

An additional option, if you have FOCUS on your PC, is to create an extract file with FOCUS and send it to your PC where you create the actual graph.

Medium-Resolution Devices

These devices use step motors to drive platens back and forth across the pages to draw two series of spaced dots that simulate continuous lines. There are separate device symbols for the most frequently used printers (see DEVICE in *SET Parameters* on page 19-59), and a generic device code, HIGHRES (or HIGHRS12 for 12 pitch), for use with many unlisted printers.

Pie charts are not available on these devices.

When using this type of printer, set PAUSE=ON so that you can adjust the paper in the printer before drawing the graph.

High-Resolution Devices

This section describes the special considerations that apply when directing your FOCUS graphs to high-resolution devices from IBM, Hewlett-Packard, and Tektronix.

IBM Devices Using GDDM

To produce graphs on IBM graphics printers or high-resolution graphics terminals you must have IBM's Graphical Data Display Manager (GDDM). GDDM provides various subroutines for saving, printing, and copying graphic screen contents. FOCUS produces graphs on IBM terminals or printers when you set DEVICE=IBM3279.

See your IBM representative concerning the proper configuration for your device controller and terminal.

GDDM Default Conditions

Whenever graphs are created using FOCUS and GDDM, the printed form of the graph (activated by pressing the PF4 key) has a default size of 132 by 80 characters on 3284 or 3287 printers. These sizes are independent of the parameters that control the lengths of the axes. As a default, each graph is presented with a frame (border). If you wish to omit the frame, set FRAME=OFF.

GDDM Save and Print Facilities

GDDM includes facilities for saving generated graphs that you activate by pressing the PF1 key to save graphs in an ADMSAVE file on your operating system. Thus saved, you can subsequently use the IBM program ADMUSF2 (supplied with GDDM) to display the saved screens.

For special instructions covering the positioning of graphs on IBM 3284 or 3287 printers, please refer to Technical Memorandum 7689, *Plot Table Settings* (available through your Information Builders Branch Office).

Graphics Device Characteristics

To draw vectors, use 7-color displays, or define your own special field patterns, you need a 3279 Model 2B, 3B, 3SG, or 3X with a 3274 terminal controller and C configuration support. C supports structured field and attribute processing (SFAP) and the use of programmed symbols (PS). (The Model 3276 terminal controller does not use C.)

3279 Models 2A and 3A have only Base Color which automatically maps colors to preset 3270 field types:

- Protected intensified becomes white.
- Unprotected intensified becomes red.
- Protected normal intensity fields become blue.
- Unprotected normal intensity fields become green.

Thus, FIDEL is automatically color coded with no programming changes, but only in a base color. Additional colors are available with the 3SG, 3X and the older B models.

Hewlett-Packard Plotters

The Hewlett-Packard 7220 series plotters translate FOCUS graph requests into 4- or 8-color graphs, suitable for presentations. Color selections and assignments are made using the standard Hewlett-Packard procedures. (Special pens are available from Hewlett-Packard for plotting graphs on transparencies for overhead projection.) For plotters with optional text facilities FOCUS has special parameters for controlling:

- Text positioning (column, line, and spacing).
- Color pen selection (red, blue, green, black).
- Letter sizes (two or four times the default size).
- Special font selection (slanted text).

To activate Hewlett-Packard plotters use the appropriate form of the SET TERM or DEVICE (see *SET Parameters* on page 19-59). FOCUS provides default axes lengths and scaling, but these and the other graphic elements can all be changed by adjusting SET parameters discussed in *Adjusting Graph Elements* on page 19-38 and summarized in *SET Parameters* on page 19-59.

Ordinarily, plotters are connected in line with a terminal and a modem. Thus, you can refine your graph requests, viewing the output on the terminal, until you produce exactly what you want and then set the DEVICE parameter to your plotter and issue the REPLOT command to produce the hard copy.

Use the plotter controls to position graphs anywhere on sheets of paper up to 11 by 16.5 inches. Unless you change the default paper size, FOCUS prepares output for an 8.5 by 11 inch sheet placed lengthwise in the lower left-hand corner of the plotter. The other default assignments are as follows:

```
HAXIS=130, VAXIS=66, GCOLOR=ON
```

Tektronix Color Terminals

Tektronix high-resolution CRTs can display the output from GRAPH requests, but only in black and white. The sizes of the vertical and horizontal axes are set depending on the device selected and cannot be overridden. Select the appropriate device number from those listed in *SET Parameters* on page 19-59.

Command and SET Parameter Summary

The FOCUS GRAPH command plots data retrieved with request statements in the form of a graph, with horizontal and vertical axes. Many of the elements used in TABLE requests are used in exactly the same way in GRAPH requests.

The GRAPH environment also includes a set of parameters that control the look of the graph and offer some additional control at run time (for example, pause to adjust paper before printing, select a device, etc.).

GRAPH Command

In the syntax samples that follow, the elements are the same as those used in TABLE requests. The complete set is shown here but the elements are described more fully in Chapter 4, *Sorting Tabular Reports*.

Syntax How to Enter the Environment

To enter the GRAPH environment enter the following:

```
GRAPH FILE filename
```

Syntax How to Specify Annotating Text

Heading strings can contain any character except the double quotation mark (“), and can also contain field references and formatting controls.

Heading—This syntax is used to specify graph headings:

```
[HEADING [CENTER]]
"string1"
["string2"]
```

Field reference format—This syntax is used to specify field reference format:

```
<[prefix.]fieldname[>]
```

Formatting Controls—The following formatting controls may be specified as part of a graph request:

```
Tab to column "n" . . . . . . . . . .<n
Tab "n" columns to the right . . . .<+n
Tab "n" columns to the left . . . . <-n
Return to column 1
   and advance "n" lines. . . . . . .</n
Name a color for a line . . . . . . <.color
Select special font
 [BIG, SLANT or BLOCK on HP7220]. .<.fontname
 ("BIG" doubles the character
  sizes, "BLOCK" quadruples them)
Reset controls to default settings <.CLEar
```

Syntax How to Name the Subject and Graph Type

The following syntax is used to specify the subject and graph type:

```
{command}  object1    [[AND|OVER] object2...object5 ]
```

where:

command

Is one of the following: PRINT, WRITE, SUM, ADD or COUNT.

Syntax How to Specify Display Fields

Display fields can be any of the following:

```
[prefix.]fieldname [AS 'string'] [IN position]
COMPUTE name1 [/format1] = expression1;[AS 'string1']
COMPUTE name2 [/format2] = expression2;[AS 'string2']
```

Syntax How to Specify Horizontal Sorting of Data Points

The following syntax is used for horizontal sorting of data:

```
ACROSS fieldname  [IN-GROUPS-OF n [TOP]][AS 'string']
ACROSS fieldname  [IN position]
```

Syntax How to Specify Separate Graphs or Vertical Sorting of Plot Points

The following syntax is used for specifying separate graphs or vertical sorting of plot points:

```
BY fieldname [IN-GROUPS-OF n [TOP]][AS 'string']
```

Syntax How to Save the Formatted Graph Data in a File

The following syntax is used to save formatted graph data in a file:

```
ON [GRAPH] SAVE [AS filename] FORMAT GRAPH
```

Syntax How to Complete the GRAPH Request

To complete a graph request, type the command END on a separate line:

```
END
```

If you do not wish to complete the graph request, use one of the following methods to abort the request and return to FOCUS:

- To quit in the middle of a graph request, type the command QUIT on a separate line:

  ```
  QUIT
  ```

- To terminate the display of a graph, type the command HT from the command line:

  ```
  HT
  ```

Syntax How to Concatenate Unlike Data Sources

To concatenate unlike data sources in a single graph request, divide your request into one main request that retrieves the first file and a subrequest for each concatenated file. The main request defines the data fields, sorting criteria, and output format for each file. The MORE command concatenates each file after the first. The syntax is:

```
GRAPH FILE file1
   main request
MORE
FILE file2
   subrequest
 MORE
    .
    .
    .
END
```

Note: IF and WHERE selection tests apply only to the subrequest in which they appear.

SET Parameters

To set the parameters which control the GRAPH environment use the appropriate variation of the SET command. The syntax is as follows:

```
SET parameter=value,parameter=value...
```

For example:

```
SET HAXIS=75,VAXIS=40
SET GRID=OFF,BARSPACE=2,BARWIDTH=3
```

Note:

- Repeat the command SET on each new line.
- When entering more than one parameter on a line, separate them with commas.
- You can use unique truncations of parameter names. You must make sure that they are unique.

To review the current parameter settings, issue the command

```
? SET GRAPH
```

which produces a listing of the values.

The table that follows lists all of the parameters in alphabetic sequence, showing the name, range of values (default is underlined), and function of each.

Parameter Name	Range of Values	Parameter Function
AUTOTICK	ON/OFF	When ON, FOCUS automatically sets the tick mark intervals. (See also HTICK and VTICK.)
BARNUMB	ON/OFF	Places the summary values at the ends of the bars on bar charts, or slices on pie charts.
BARSPACE	0-20	Specifies the number of lines separating the bars on bar charts.
BARWIDTH	1-20	Specifies the number of lines per bar on bar charts.
BSTACK	ON/OFF	Specifies that the bars on a bar chart are to be stacked rather than placed side by side.
DEVICE or TERMINAL	IBM3270	Specifies the plotting device or terminal to be used. When the default is used, low-resolution graphics are sent to your terminal or to the printer if PRINT=OFFLINE (see the SET command in the *Developing Applications* manual). Medium- and high-resolution devices are selected by entering one of the following parameter settings for the DEVICE (or TERMINAL).
Medium-resolution devices:		
	AJ	Specifies Anderson Jacobson - Model AJ830.
	AJ12	Specifies Anderson Jacobson - Model AJ832 (12 Pitch).
	DIABLO	Specifies Diablo - Model 1620.
	DIABLO12	Specifies Diablo - Model 1620 (12 Pitch).
	GS	Specifies Gencom.
	GS12	Specifies Gencom (12 Pitch).
	HIGHRES	Specifies generic device for most medium-resolution graphic devices.
	HIGHRS12	Specifies generic device -see above (12 Pitch).
	TRENDATA	Specifies Trendata - Model 4000A.
	TRENDT12	Specifies Trendata - Model 4000A (12 Pitch).

Parameter Name	Range of Values	Parameter Function
High-resolution devices from Hewlett-Packard:		
	`HP7220`	Specifies HP Models 7229A and 7470A. Both are 4-pen plotters with no paper advance. Model 7470 requires a special Y cable (Part #17455).
	`HP7220S`	Specifies HP Model 7220S, 4-pen plotter with paper advance.
	`HP7220C`	Specifies HP Models 7220C and 7475A. Both are 8-pen plotters with no paper advance. Model 7475 requires a special Y cable (Part #17455).
	`HP7220T`	Specifies HP Model 7220T, 8-pen plotter with paper advance.
	`HP7221`	Specifies HP Model 7221, 4-pen plotter with no paper advance.
	`HP7221S`	Specifies HP Model 7221S, 4-pen plotter with paper advance.
	`HP7221C`	Specifies HP Model 7221C, 8-pen plotter with no paper advance.
	`HP7221T`	Specifies HP Model 7221T, 8-pen plotter with paper advance.

Note: The default horizontal and vertical axes for all Hewlett-Packard devices are as follows:

`HAXIS=100, VAXIS=50.`

High-resolution devices from IBM:		
	`IBM3279`	Specifies one of the following devices: Any IBM 3270 series device that supports GDDM graphics, such as the 3279-S3G, 3179, or 3472. This includes PCs with fully compatible 3270 series hardware and software. Printers: Any IBM 3270 series printer that supports GDDM graphics such as the 3287-2C and the 4224.

Note: IBM’s Graphical Data Display Manager (GDDM) is required for all of these devices. Entering this value automatically sets the following GRAPH parameter values: HAXIS=80, VAXIS=32, and GCOLOR=ON. For any monochrome device, you should set GCOLOR=OFF; for any Model 2 device (24x80 screen), you should set VAXIS=-24.

High-resolution devices from Tektronix:		
	TEK4010	Specifies one of the following models: 4010, 4050 series and 4100 series (B/W only). Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF.
	TEK4012	Specifies Model 4012. Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF.
	TEK4013	Specifies Model 4013. Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF.
	TEK4014	Specifies Model 4014. Automatically sets HAXIS=133, VAXIS=64, and GCOLOR=OFF.
	TEK4014E	Specifies Model 4014E. Automatically sets HAXIS=133, VAXIS=64, and GCOLOR=OFF.
	TEK4015	Specifies Model 4015. Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF.
	TEK4015E	Specifies Model 4015E. Automatically sets HAXIS=74, VAXIS=35, and GCOLOR=OFF.
	TEK4025	Specifies Model 4025. Automatically sets HAXIS=80, VAXIS=32, and GCOLOR=OFF.
	TEK4027	Specifies Model 4027. Automatically sets HAXIS=80, VAXIS=32, and GCOLOR=ON.
	TEK4662	Specifies Model 4662. Plot address is D. It is recommended that you set GCOLOR=OFF, HAXIS=80, VAXIS=32. If the Model 4662 is connected to a Model 4025, set DEVICE=TEK4025. If the Model 4662 is connected to a Model 4027, set DEVICE=TEK4027.

Parameter Name	**Range of Values**	**Parameter Function**
FRAME	ON/OFF	For GDDM graphics, ON (the default) indicates you want a frame around your graph. To omit the Frame, set OFF.
GCOLOR (or GRIBBON)	ON/OFF	On medium- and high-resolution devices, setting this parameter OFF causes different black and white patterns to be substituted for colors. On medium-resolution devices, setting it ON causes alternation between black and red ribbons on multiline plots. **Note:** 3287 printers use black, red, blue, and green.
GMISSING	ON/OFF	If ON, specifies that variables with the value specified in GMISSVAL are to be ignored.
GMISSVAL	*nn*	Specifies the variable value that represents missing data.
GPROMPT	ON/OFF	When ON, FOCUS prompts for all pertinent graph parameters.
GRIBBON		See GCOLOR.
GRID	ON/OFF	When ON, specifies that a grid of parallel horizontal lines is to be drawn on the graph at the vertical class marks (see also VGRID).
GTREND	ON/OFF	When ON, specifies that a trend line is to appear on scatter diagrams.
HAUTO	ON/OFF	Specifies automatic scaling of the horizontal axis unless overridden by the user. If OFF, user must supply values for HMAX and HMIN.
HAXIS	20-130	Specifies the width in characters of the horizontal axis. This parameter can be adjusted for graphs generated OFFLINE. HAXIS is ignored for ONLINE displays since FOCUS automatically adjusts the width of the graph to the width of the terminal.
HCLASS	*nnn*	Specifies the horizontal class interval when AUTOTICK=OFF.
HISTOGRAM	ON/OFF	When ON, FOCUS draws a histogram instead of a curve when values on the horizontal axis are not numeric.

Parameter Name	Range of Values	Parameter Function
HMAX	*nnn*	Specifies the maximum value on the horizontal axis when the automatic scaling is not used (HAUTO=OFF).
HMIN	*nnn*	Specifies the minimum value on the horizontal axis when the automatic scaling is not used (HAUTO=OFF).
HSTACK	ON/OFF	Specifies that the bars on a histogram are to be stacked rather than placed side by side.
HTICK	*nnn*	Specifies the horizontal axis tick mark interval, when AUTOTICK is OFF.
PAUSE	ON/OFF	Specifies whether there will be a pause for paper adjustment on the plotter after the request is executed.
PIE	ON/OFF	Specifies a pie chart is desired (only available on high-resolution devices).
PLOT		Specifies the width and height settings for a graphic printer if DEVICE=IBM3279 or HIGHRES. Hexadecimal values must be supplied. For example `SET PLOT=0050,0018` produces a printed plot 80 by 24 decimal characters (50 hex = 80 decimal, 18 hex = 24 decimal). When used, the PLOT parameter must be the last parameter set.
PRINT	ONLINE/OFFLINE	When OFFLINE is entered, the graph is printed on the system high-speed printer.
TERM		See DEVICE.
VAUTO	ON/OFF	Specifies automatic scaling of the vertical axis unless overridden by the user. If OFF, user must supply values for VMAX and VMIN.

Parameter Name	Range of Values	Parameter Function
`VAXIS`	`20-66`	Page length in lines. This parameter can be adjusted for graphs generated OFFLINE. VAXIS is ignored for ONLINE displays since FOCUS automatically adjusts the height of the graph to the height of the terminal.
`VCLASS`	*nnn*	Specifies the vertical class interval when AUTOTICK=OFF.
`VGRID`	`ON/OFF`	When ON, specifies that a grid of parallel vertical lines is to be drawn on the graph at the horizontal class marks (see also GRID).
`VMAX`	*nnn*	Specifies the maximum value on the vertical axis when the automatic scaling is not used (VAUTO=OFF).
`VMIN`	*nnn*	Specifies the minimum value on the vertical axis when the automatic scaling is not used (VAUTO=OFF).
`VTICK`	*nnn*	Specifies the vertical axis tick mark interval, when AUTOTICK is OFF.
`VZERO`	`ON/OFF`	Normally missing values on the vertical axis are ignored (VZERO=OFF). If ON, the values are treated as zero (0).

20 Using SQL to Create Reports

Topics:

- Supported and Unsupported SQL Statements
- Using the SQL Command
- Index Optimized Retrieval
- TABLEF Optimization

FOCUS provides the SQL Translator, which translates ANSI Level 2 SQL statements into executable FOCUS requests. This command was designed for SQL users with limited FOCUS knowledge who want to employ the power of FOCUS report formatting. SQL users can create procedures that enable them to format output quickly and easily by combining their knowledge of basic FOCUS TABLE commands and SQL.

Note: Because the SQL Translator is ANSI Level 2 compliant, some requests that worked in prior releases may no longer work.

Supported and Unsupported SQL Statements

SQL Translation Services is ANSI Level 2 compliant. This section contains a summary of supported and unsupported SQL statements and the expected results from SQL queries.

Supported SQL Statements

SQL Translation Services supports the following:

- SELECT including SELECT ALL and SELECT DISTINCT.
- CREATE TABLE. The following data types are supported for CREATE TABLE: REAL, DOUBLE PRECISION, FLOAT, INTEGER, DECIMAL, CHARACTER, and SMALLINT.
- INSERT, UPDATE, and DELETE for relational data sources, IMS, and FOCUS files.
- Equijoins and non-equijoins.
- CREATE VIEW and DROP VIEW.
- PREPARE and EXECUTE.
- Delimited identifiers of table names and column names. Table and column names containing embedded blanks or other special characters in the SELECT list should be enclosed in double quotation marks.

- Column names qualified by table names or by table tags.
- The UNION and UNION ALL operators.
- Non-correlated subqueries for all requests.
- Correlated subqueries for requests that are candidates for Dialect Translation to an RDBMS that supports this feature.
- Numeric constants, literals, and expressions of literals in the SELECT list.
- The following scalar functions for all queries: DECIMAL, FLOAT, INTEGER, and SUBSTR (or SUBSTRING).
- The following scalar functions for queries that are candidates for Dialect Translation if the RDBMS engine supports the scalar function type: CHAR, DATE, DAY, DAYS, DIGITS, HEX, HOUR, LENGTH, MICROSECOND, MINUTE, MONTH, SECOND, TIME, TIMESTAMP, VALUE, VARGRAPHIC, and YEAR.
- The concatenation operator, `'`||,`'` or the syntax `'`*`alpha1`*` CONCAT `*`alpha2`*`'` used with literals or alphanumeric columns.
- The following aggregate functions: COUNT, MIN, MAX, SUM, and AVG.
- Date literals of formats YYYY-MM-DD, YYYY/MM/DD, MM-DD-YYYY, and MM/DD/YYYY.
- All requests that contain ANY, SOME, and ALL that do not contain =ALL, <>ANY, and <>SOME.
- =ALL, <>ANY, and <>SOME for requests that are candidates for Dialect Translation if the RDBMS engine supports quantified subqueries.
- The special registers USER, CURRENT DATE, CURRENT TIME, CURRENT TIMESTAMP, CURRENT EDASQLVERSION, and CURRENT TIMEZONE.
- NULL and NOT NULL predicates.
- LIKE and NOT LIKE predicates.
- IN and NOT IN predicates.
- EXISTS and NOT EXISTS predicates.
- GROUP BY clause expressed using explicit column names.
- ORDER BY clause expressed using explicit column names or column numbers. Note that ORDER BY with UNION supports the column number syntax only.
- FOR FETCH ONLY feature to circumvent record locking.
- Continental Decimal Notation when the CDN variable is set.
- National language support.

Unsupported SQL Statements

SQL Translation Services does not support the following:

- More than 15 joins per SELECT. This is a SQL limit. FOCUS supports up to 16 joins.
- Outer joins (that is, FOCUS ‘ALL = ON’ or ‘ALL=PASS’ is not supported).
- ALIAS names in Master Files and the use of formatting options to format output.
- Unique truncations of column names.
- Temporary defined columns. Permanent defined columns are supported (that is, those that are defined in the EDA Dynamic Catalog or FOCUS Master File).
- Correlated subqueries in DML Generation.
- =ALL, <>ANY, and <>SOME in DML Generation.
- Date arithmetic.

Expected Results From SQL Queries

All SQL queries produce Cartesian product style answer sets. In addition, all dates are converted to YYYYMMDD format on output regardless of their original format.

Reserved Words

The following words may not be used as field names in a Master File that is used with the SQL Translator:

- ALL
- COUNT
- SUM
- MAX
- MIN
- AVG
- CURRENT
- DISTINCT
- USER

Using the SQL Command

The SQL command may be used to report from any file or set of files that FOCUS can read, including FOCUS files and external files. Standard TABLE subcommands to format reports can be appended to the SQL statements to take advantage of a wide range of FOCUS report preparation options.

If you need to join files for your request, you have two options. You can use the FOCUS JOIN command before you issue any SQL statements, or you may use the WHERE clause in the SQL SELECT statement to join the required files dynamically.

Syntax How to Use the SQL Commands

The SQL command has the following syntax:

```
SQL
sql statement;
[options]
[TABLE subcommands]
END
```

The SQL command consists of the following components, listed in the order in which they must appear:

- The SQL command identifier, which invokes the translator.
- An SQL statement. The statement must be terminated by a semicolon; it can continue for more than one line.
- Two debugging options, ECHO ON and FILE, which capture the generated TABLE request. These options are placed after the SQL statement:

  `ECHO [ON]`

  Displays the resulting TABLE request on the terminal.

  `FILE [name]`

  Writes the translated FOCUS commands to the named FOCEXEC file. The default name is FOCSQL99 if no file name is supplied. This file, FOCSQL99, is temporary and exists only for the duration of its execution.
- Optional FOCUS TABLE formatting commands appearing after the header TABLE.
- END or QUIT.

Note: The following guidelines apply to the specification of syntax for the SQL Translator:

- Field names are limited to 48 characters, an ANSI standard Level 2 limitation.
- View names, generated through the SQL CREATE VIEW statement, are limited to 18 characters.

Example Using the SQL Command

The following example shows an SQL request based on the previous discussion:

```
SQL
SELECT BODYTYPE, AVG(MPG), SUM(SALES)
FROM CAR
WHERE RETAIL_COST > 5000
GROUP BY BODYTYPE;
TABLE HEADING CENTER
"AVERAGE MPG AND TOTAL SALES PER BODYTYPE"
END
```

Reference TABLE Formatting Subcommands

TABLE formatting subcommands may be used in a request, subject to the following rules:

- You must include the keyword TABLE at the beginning of the first line of FOCUS subcommands. (This is a keyword, not a TABLE FILE file name command.)
- You may specify headings and footings, describe actions with an ON phrase, or use the ON TABLE SET command. Additionally, you can use ON TABLE HOLD or ON TABLE PCHOLD to create an extract file. You can also specify READLIMIT and RECORDLIMIT tests.
- You cannot specify additional verb objects, ACROSS fields, WHERE or IF statements (other than READLIMIT or RECORDLIMIT tests), or COMPUTEs; BY clauses are ignored.
- You can use TABLE formatting commands with SELECT and UNION only.

The SELECT Statement

The SQL SELECT statement translates into one or more FOCUS TABLE or TABLEF PRINT or WRITE commands, depending on whether aggregation is applied in the request.

The SQL statement SELECT * translates to a PRINT of every field in the Master File and uses all of the fields of the Cartesian product. This is a quick way to display a file, provided it fits in a reasonable number of screens for display, or provided you use ON TABLE HOLD or ON TABLE PCHOLD.

SQL functions (such as COUNT, SUM, MAX, MIN, AVG) are supported in SELECT lists and HAVING conditions. Expressions may be used as function arguments.

The function COUNT (*) translates to a count of the number of records produced by printing all fields in the Master File. This is the same as counting all rows in the Cartesian product that results from a SELECT on all fields.

Expressions in the SQL WHERE predicate are translated into corresponding expressions in FOCUS WHERE clauses, whenever possible. Expressions in SELECT lists generate DEFINEs. The SQL HAVING and BETWEEN conditions also translate into corresponding FOCUS WHERE clauses. The SQL LIKE condition is translated directly into the corresponding FOCUS LIKE condition in the WHERE clause.

FOCUS only supports subqueries based on equality when the WHERE expression is compared to a subquery using an equal (=) sign. For example: WHERE field = (SELECT …).

Note: Correlated subqueries are not supported.

The SQL UNION operator translates to a TABLE request that makes a HOLD file for each file specified, followed by a MATCH command with option HOLD OLD-OR-NEW.

Joins

When performing SQL joins, FOCUS rules are followed and the formats of the joined fields must be the same. Join fields need not be indexed, and non-equijoins are supported.

Joined fields in the WHERE clause must be qualified if the names are not unique; that is, they must be specified with their corresponding file names (or file aliases). In the following example, T.A and U.B are qualified field names:

```
SQL
 SELECT A, B
 FROM T, U
 WHERE T.A = U.B;
END
```

Recursive joins are supported. For example, in the following statement, A and B are aliases for the same file, CAR. The result from file CAR is pairs of B values having the same A value:

```
SQL
    SELECT A.SEATS, B.SEATS
    FROM CAR A, CAR B
    WHERE A.MODEL = B.MODEL;
END
```

All field names in the SELECT clause must be unique, or field qualification must be used.

Joins issued by the SQL Translator are assigned names in the format,

`SQLJNM`*`nn`*

where:

`SQLJNM`

Is the SQL Translator join prefix.

`nn`

Is a number between 01 and 16 assigned by FOCUS in the order in which the joins are created. (FOCUS supports up to a maximum of 16 joins.)

For example, the first join will have an AS name of SQLJNM01, the second join will be named SQLJNM02, up to SQLJNM16.

All joins are automatically created and cleared by the SQL Translator. No user-specified joins are affected.

Reference Usage Notes for Joining Fields

- In standard SQL, WHERE field=‘a’ selects records where the field has the value ‘a’ or ‘A’. The SQL Translator is case sensitive and returns only the exact value requested (‘a’ only).
- The SQL comparison operators ANY, SOME, and ALL are supported, with the exception of =ALL, <>ANY, and <>SOME.
- Sub-selects are not supported in HAVING conditions.
- Escape characters are not supported in the LIKE predicate.
- In a multi-segment structure, parent segments are omitted from reports if no instances of their descendant segments exist. This is an inner join.
- The SQL Translator applies optimization techniques when constructing joins. These are described in *Index Optimized Retrieval* on page 20-14.

For information about index optimization and optimized Join statements, see the EDA documentation, or *Index Optimized Retrieval* on page 20-14.

INSERT, UPDATE, and DELETE

You can issue an SQL INSERT, UPDATE, or DELETE command against one segment instance (row) at a time. When you issue one of these commands against a multi-segment Master File:

- All fields referenced in the command must be on a single path through the file structure.
- The command must explicitly specify (in the WHERE predicate) every key value from the root to the target segment instance, and this combination of key values must uniquely identify one segment instance (row) to be affected by the command.

  If you are modifying every field in the row, you can omit the list of field names from the command.
- The SQL Translator does not support subqueries, such as:

  ```
  INSERT...INTO...SELECT...FROM...
  ```

Although each INSERT, UPDATE, or DELETE command can specify only one row, referential integrity constraints may produce the following additional modifications to the data source:

- If you delete a segment instance that has descendant segment instances (children), the children are automatically deleted.
- If you insert a segment for which parent segments are missing, the parent segments are automatically created.

CREATE TABLE

CREATE TABLE and INSERT INTO table are supported with the SQL Translator. This enables you to create tables to enhance reporting efficiency.

Note: CREATE TABLE only generates single-segment Master Files.

Single-record insert with actual data values is supported. The following example shows the use of the single-record insert, creating the table U with 1 record:

```
-* Single-record insert example.
-*
SQL
CREATE TABLE U (A INT, B CHAR(6), C CHAR(6), X INT, Y INT);
END
SQL
INSERT INTO U (A,B,C,X,Y) VALUES (10, '123456','654321', 10, 15);
END
```

According to normal SQL data definition syntax, each CREATE TABLE or INSERT statement must terminate with a semicolon.

When using the CREATE TABLE and INSERT commands, the datatype FLOAT should be declared with a precision and used in an INSERT without the ‘E’ designation. This requires the entire value to be specified without an exponent. For example, to insert the values 100.00E01 and 200.00E01, you would specify:

```
SQL
CREATE TABLE T (A FLOAT(6), B CHAR(4))
END
SQL
INSERT INTO T (A,B) VALUES (1000.00,'1234');
END
SQL
INSERT INTO T (A,B) VALUES (2000.00,'4321');
END
```

Reference Usage Notes for CREATE TABLE

- The CREATE TABLE command supports the INTEGER, SMALLINT, FLOAT, CHARACTER, DECIMAL, DOUBLE PRECISION and REAL datatypes. Decimals are rounded in the DOUBLE PRECISION and REAL datatypes.
- The CHECK and DEFAULT options are not supported with the CREATE TABLE command.
- CREATE TABLE and INSERT need not be used together.

CREATE VIEW and DROP VIEW

The SQL statement

```
CREATE VIEW viewname AS subquery ;
```

is supported. To use the view, issue a SELECT from the view. You cannot issue a TABLE request against the view, because the view is not held as a physical FOCUS file. To create a HOLD file, specify ON TABLE HOLD after the SQL statements.

The following example creates a view named XYZ:

```
SQL
CREATE VIEW XYZ
 AS SELECT CAR, MODEL
 FROM CAR;
END
```

To report from the view, issue:

```
SQL
 SELECT CAR, MODEL
 FROM XYZ;
END
```

According to normal SQL data definition syntax, each CREATE VIEW statement must terminate with a semicolon.

Reference Usage Notes for SQL Views

- CREATE VIEW does not put the view in the system catalog.
- The DROP VIEW statement is supported. For example:

  ```
  SQL
   DROP VIEW XYZ;
  END
  ```

PREPARE, EXECUTE, and COMMIT

The SQL PREPARE statement creates a machine language form of an SQL statement and associates it with an identifier; you can then EXECUTE the SQL statement any number of times by referring to the identifier. The following example executes an SQL SELECT statement by referencing the identifier PREP_QUERY:

```
SQL
 PREPARE PREP_QUERY FROM
 SELECT COUNTRY
 FROM CAR
 WHERE LENGTH > ? AND COUNTRY = ?;
END
SQL
 EXECUTE PREP_QUERY USING 165, 'ITALY';
END
SQL
 EXECUTE PREP_QUERY USING 190, 'ENGLAND';
END
SQL
 EXECUTE PREP_QUERY USING 182, 'FRANCE';
END
SQL
 COMMIT;
END
```

In the SELECT statement, each question mark (?) indicates where you should substitute a value. You provide the necessary values in the USING clause of the EXECUTE statement in the order of the question marks in the original statement.

The COMMIT statement makes updates or inserts permanent, and clears all PREPAREd statements.

Cartesian Product Style Answer Sets

The SQL Translator automatically generates Cartesian product style answer sets unless you explicitly turn this feature off with the SET CARTESIAN command discussed in Chapter 15, *Merging Data Files*. However, this is not advised since it does not comply with ANSI standards.

Continental Decimal Notation (CDN)

Continental decimal notation displays numbers using a comma to mark the decimal position and periods for separating significant digits into groups of three. This notation is available for SQL Translator requests, as illustrated by the following example:

```
SET CDN=ON
SQL
    SELECT SEATS + 1,2
    FROM CAR;
END
```

This example creates a column defined as “1.2 + SEATS.”

Delimited Identifiers

A field name may contain (but not begin with) the symbols ., #, @, _, and $. Enclose such field names in double quotation marks when referring to them in requests. For example:

```
"COUNTRY.NAME"
```

Note: In CMS, # is the end-of-line indicator. You cannot use it in a field name that will be entered at the FOCUS prompt.

Field Names With Embedded Blanks and Special Characters

A SELECT list can specify field names with embedded blanks or other special characters; you must enclose such field names in double quotation marks. Special characters are any characters not listed in *Delimited Identifiers* and not contained in the national character set of the FOCUS environment installed.

Qualified Field Names

You can qualify a field name with file and file alias names. File alias names are described in the discussion of joins in *Joins* on page 20-7. See the *Describing Data* manual for more information on qualified field names.

For example, to qualify the delimited field name COUNTRY.NAME with its file name, use:

```
CAR."COUNTRY.NAME"
```

UNION

The SQL UNION operator generates MATCH in FOCUS. Therefore, the number of files that can participate in a UNION is determined by the MATCH limit. UNION with parentheses is supported.

Subqueries

You can nest subqueries up to 15 levels deep. Correlated subqueries are not supported.

Numeric Constants, Literals, Expressions of Literals, and Functions

The SELECT list, WHERE clause, and HAVING clause can include numeric constants, literals enclosed in single quotation marks, expressions of literals, and the following scalar functions:

- DECIMAL
- INTEGER
- SUBSTR

Internally, FOCUS creates a DEFINE field for each such item in the list, and provides the value of the DEFINE field in the answer set.

Date Formats

All date formats for actual and DEFINE fields in the Master File are converted to the form YYYYMMDD. If you specify a format that lacks any component, the SQL Translator supplies a default value for the missing component.

To specify a portion of a date, such as the month, use a DEFINE field with an alphanumeric format.

Index Optimized Retrieval

The SQL Translator improves FOCUS query performance by generating optimized code that enables the underlying retrieval engine to access the selected records directly, without scanning all segment instances.

For more information about index optimization and optimized Join statements, see the EDA documentation for your platform.

Optimized Joins

The SQL Translator accepts joins in SQL syntax. SQL language joins have no implied direction; that is, the concepts of host and cross-referenced files do not exist in SQL.

The SQL Translator analyzes each join in order to identify an efficient implementation. First, it assigns costs to the candidate joins in the query:

- Cost = 1 for an equijoin to a field that can participate as a cross-referenced field according to FOCUS join rules. This is common in queries against relational tables with equijoin predicates in the WHERE clause.
- Cost = 16 for an equijoin to a field that cannot participate as a cross-referenced field according to FOCUS join rules.
- Cost = 256 for a non-equijoin or an unrestricted Cartesian product.

The Translator then uses these costs to build a join structure for the query. The order of the tables listed in the FROM clause of the query influences the first two phases of the join analysis:

1. If there are cost=1 joins from the first table referenced in the FROM clause to the second, from the second table to the third, and so on, the Translator joins the tables in the order specified in the query. If not, it goes on to Phase 2.
2. If Phase 1 fails to generate an acceptable join structure, the Translator attempts to generate a join structure without joining any table TO a table that precedes it in the FROM clause. Therefore, this phase always makes the first table referenced in the query the host table. If there is no cost=1 join between two tables, or if using one would require changing the table order, the Translator abandons Phase 2 and implements Phase 3.
3. The Translator generates the join structure by using the lowest cost joins first and then picking from the more expensive ones as necessary. This sorting process may change the order in which tables are joined. The efficiency of the join that this procedure generates is somewhat dependent on the relative sizes of the tables being joined.

If the analysis results in joining TO a table that cannot participate as a cross-referenced file according to FOCUS rules (because it lacks an index, for example), the Translator generates code to build an indexed HOLD file and implements the join with this file. However, the HOLD file does not participate in the analysis of join order.

TABLEF Optimization

To improve performance, the SQL Translator can be set to generate FOCUS TABLEF commands instead of TABLE commands. This optimization is available via the use of the SET SQLTOPTTF command (SQL Translator OPTimization TableF). The syntax is

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

where:

`ON`

Causes TABLEF commands to be generated when possible (for example, if there is no join or GROUP BY phrase). This is the default.

`OFF`

Causes TABLE commands to be generated.

A Master Files and Diagrams

Topics:

- The EMPLOYEE Database
- The JOBFILE Database
- The EDUCFILE Database
- The SALES Database
- The PROD Database
- The CAR Database
- The LEDGER Database
- The FINANCE Database
- The REGION Database
- The COURSES Database
- The EMPDATA Database
- The EXPERSON Database
- The TRAINING Database
- The PAYHIST File
- The COMASTER File
- VideoTrk and Movies Databases

You can create these files on your user ID by executing the FOCEXEC procedures specified below. These FOCEXECs were supplied for your installation with FOCUS. If they are not available to you or if they produce error messages, contact your systems administrator.

To create these files, first make sure you have read access to the Master Files, then perform the following:

- To create the EMPLOYEE, EDUCFILE, and JOBFILE databases, under CMS enter:

  ```
  EX EMPTEST
  ```

  Under MVS, enter:

  ```
  EX EMPTSO
  ```

  These FOCEXECs also test the databases by generating sample reports. If you are using Hot Screen, remember to press either Enter or the PF3 key after each report. If the EMPLOYEE, EDUCFILE, and JOBFILE databases already exist on your user ID, the FOCEXEC will replace the databases with new copies. This FOCEXEC assumes that the high-level qualifier for the FOCUS databases will be the same as the high-level qualifier for the MASTER PDS that was unloaded from the tape.

- To create the SALES and PROD databases, enter:

  ```
  EX SALES
  EX PROD
  ```

  This FOCEXEC will only create the SALES and PROD databases if they do not already exist; it will not replace a database. To replace a database, first erase it (CMS ERASE, TSO DELETE, or DYNAM DELETE), then execute GSTART.

- The CAR database is created automatically during the installation process.

- To create the LEDGER, FINANCE, REGION, COURSES, and EXPERSON databases, execute the appropriate FOCEXEC. For example, to create the LEDGER database, enter:

  ```
  EX LEDGER
  ```

- To create the EMPDATA and TRAINING databases, execute the LOADEMP and LOADTRAI FOCEXECs, respectively.

- The corresponding file for the PAYHIST Master File is called PAYHIST DATA. It is a sequential file and is allocated during the installation process.

- A corresponding FOCEXEC does not exist for the COMASTER database, because COMASTER is used for debugging other Master Files.

- To create the VideoTrk and Movies databases issue:

  ```
  EX LOADVTRK
  ```

The EMPLOYEE Database

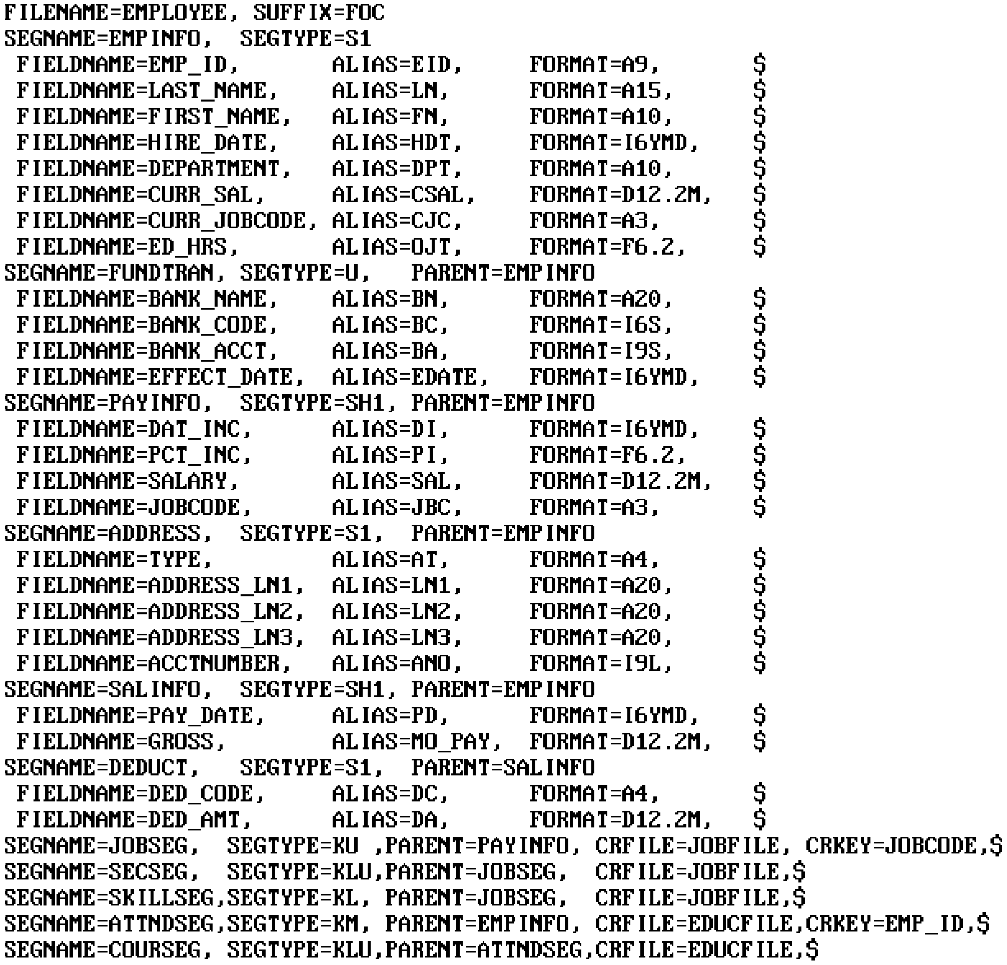
The EMPLOYEE database contains data about a company’s employees. Its segments are:

- EMPINFO, which contains employee IDs, names, and positions.
- FUNDTRAN, which specifies employees’ direct deposit accounts. This segment is unique.
- PAYINFO, which contains the employee’s salary history.
- ADDRESS, which contains employees’ home and bank addresses.
- SALINFO, which contains data on employees’ monthly pay.
- DEDUCT, which contains data on monthly pay deductions.

The EMPLOYEE database also contains cross-referenced segments belonging to the JOBFILE and EDUCFILE files, described later in this appendix. The segments are:

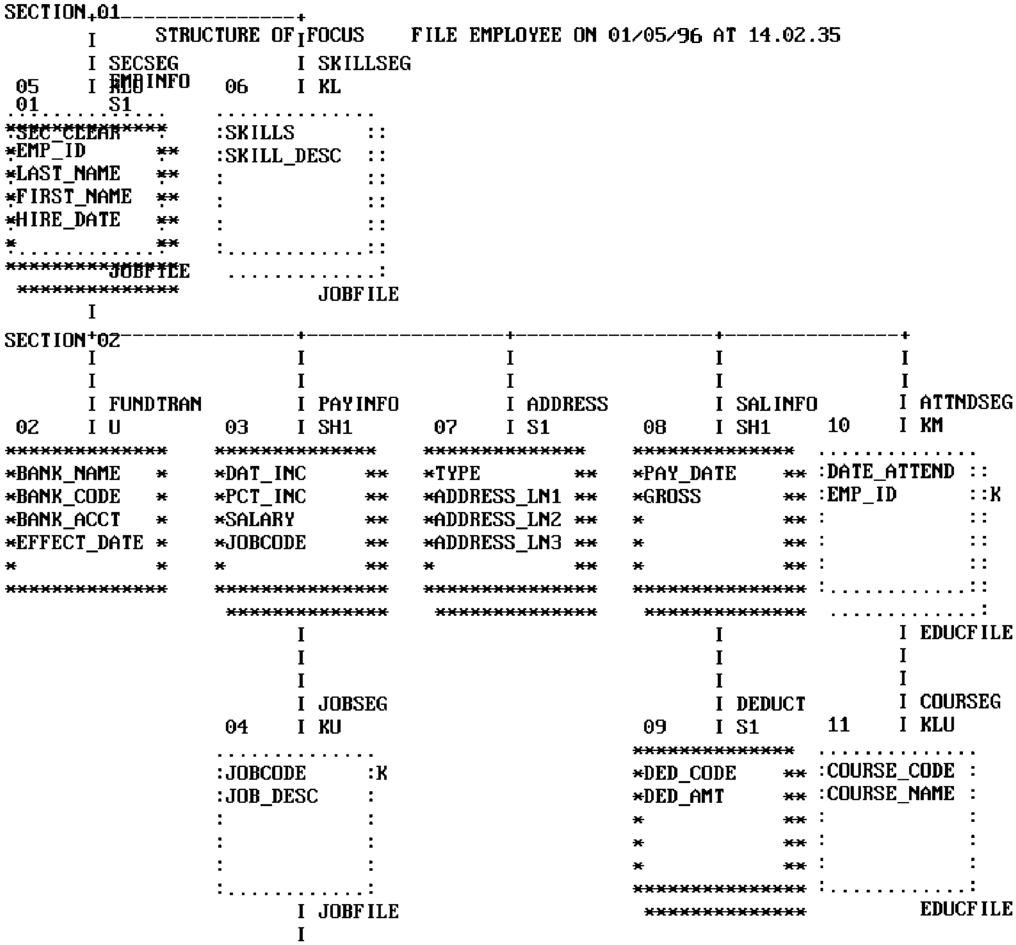
- JOBSEG (from JOBFILE), which describes the job positions held by each employee.
- SECSEG (from JOBFILE), which lists the skills required by each position.
- SKILLSEG (from JOBFILE), which specifies the security clearance needed for each job position.
- ATTNDSEG (from EDUCFILE), which lists the dates that employees attended in-house courses.
- COURSEG (from EDUCFILE), which lists the courses that the employees attended.

The EMPLOYEE Master File



```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,       ALIAS=EID,     FORMAT=A9,       $
 FIELDNAME=LAST_NAME,    ALIAS=LN,      FORMAT=A15,      $
 FIELDNAME=FIRST_NAME,   ALIAS=FN,      FORMAT=A10,      $
 FIELDNAME=HIRE_DATE,    ALIAS=HDT,     FORMAT=I6YMD,    $
 FIELDNAME=DEPARTMENT,   ALIAS=DPT,     FORMAT=A10,      $
 FIELDNAME=CURR_SAL,     ALIAS=CSAL,    FORMAT=D12.2M,   $
 FIELDNAME=CURR_JOBCODE, ALIAS=CJC,     FORMAT=A3,       $
 FIELDNAME=ED_HRS,       ALIAS=OJT,     FORMAT=F6.2,     $
SEGNAME=FUNDTRAN, SEGTYPE=U,   PARENT=EMPINFO
 FIELDNAME=BANK_NAME,    ALIAS=BN,      FORMAT=A20,      $
 FIELDNAME=BANK_CODE,    ALIAS=BC,      FORMAT=I6S,      $
 FIELDNAME=BANK_ACCT,    ALIAS=BA,      FORMAT=I9S,      $
 FIELDNAME=EFFECT_DATE,  ALIAS=EDATE,   FORMAT=I6YMD,    $
SEGNAME=PAYINFO,  SEGTYPE=SH1, PARENT=EMPINFO
 FIELDNAME=DAT_INC,      ALIAS=DI,      FORMAT=I6YMD,    $
 FIELDNAME=PCT_INC,      ALIAS=PI,      FORMAT=F6.2,     $
 FIELDNAME=SALARY,       ALIAS=SAL,     FORMAT=D12.2M,   $
 FIELDNAME=JOBCODE,      ALIAS=JBC,     FORMAT=A3,       $
SEGNAME=ADDRESS,  SEGTYPE=S1,  PARENT=EMPINFO
 FIELDNAME=TYPE,         ALIAS=AT,      FORMAT=A4,       $
 FIELDNAME=ADDRESS_LN1,  ALIAS=LN1,     FORMAT=A20,      $
 FIELDNAME=ADDRESS_LN2,  ALIAS=LN2,     FORMAT=A20,      $
 FIELDNAME=ADDRESS_LN3,  ALIAS=LN3,     FORMAT=A20,      $
 FIELDNAME=ACCTNUMBER,   ALIAS=ANO,     FORMAT=I9L,      $
SEGNAME=SALINFO,  SEGTYPE=SH1, PARENT=EMPINFO
 FIELDNAME=PAY_DATE,     ALIAS=PD,      FORMAT=I6YMD,    $
 FIELDNAME=GROSS,        ALIAS=MO_PAY,  FORMAT=D12.2M,   $
SEGNAME=DEDUCT,   SEGTYPE=S1,  PARENT=SALINFO
 FIELDNAME=DED_CODE,     ALIAS=DC,      FORMAT=A4,       $
 FIELDNAME=DED_AMT,      ALIAS=DA,      FORMAT=D12.2M,   $
SEGNAME=JOBSEG,  SEGTYPE=KU ,PARENT=PAYINFO, CRFILE=JOBFILE, CRKEY=JOBCODE,$
SEGNAME=SECSEG,  SEGTYPE=KLU,PARENT=JOBSEG,  CRFILE=JOBFILE,$
SEGNAME=SKILLSEG,SEGTYPE=KL, PARENT=JOBSEG,  CRFILE=JOBFILE,$
SEGNAME=ATTNDSEG,SEGTYPE=KM, PARENT=EMPINFO, CRFILE=EDUCFILE,CRKEY=EMP_ID,$
SEGNAME=COURSEG, SEGTYPE=KLU,PARENT=ATTNDSEG,CRFILE=EDUCFILE,$
```

The EMPLOYEE Structure Diagram



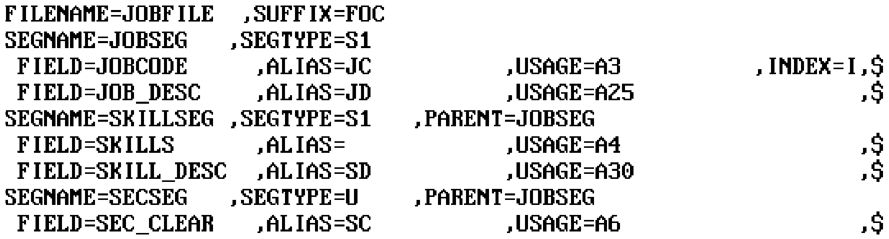
```
SECTION 01---------------+
       I     STRUCTURE OFIFOCUS    FILE EMPLOYEE ON 01/05/96 AT 14.02.35
       I SECSEG          I SKILLSEG
         EMPINFO
 05    I KLU       06    I KL
 01      S1
..............    ..............
***************
:SEC_CLEAR   :    :SKILLS      ::
*EMP_ID      **   :SKILL_DESC  ::
*LAST_NAME   **   :            ::
*FIRST_NAME  **   :            ::
*HIRE_DATE   **   :            ::
*            **   :            ::
:............:    :............::
***************
         JOBFILE   .............:
 ***************           JOBFILE
       I
SECTION 02---------------+-----------------+-----------------+---------------+
       I                 I                 I                 I               I
       I                 I                 I                 I               I
       I FUNDTRAN        I PAYINFO         I ADDRESS         I SALINFO       I ATTNDSEG
 02    I U         03    I SH1       07    I S1        08    I SH1     10    I KM
**************    **************    **************    **************  ..............
*BANK_NAME   *    *DAT_INC     **    *TYPE        **    *PAY_DATE    ** :DATE_ATTEND ::
*BANK_CODE   *    *PCT_INC     **    *ADDRESS_LN1 **    *GROSS       ** :EMP_ID      ::K
*BANK_ACCT   *    *SALARY      **    *ADDRESS_LN2 **    *            ** :            ::
*EFFECT_DATE *    *JOBCODE     **    *ADDRESS_LN3 **    *            ** :            ::
*            *    *            **    *            **    *            ** :            ::
**************    ***************   ***************   *************** :............::
                   **************    **************    **************   .............:
                         I                                   I               I EDUCFILE
                         I                                   I               I
                         I                                   I               I
                         I JOBSEG                            I DEDUCT        I COURSEG
                   04    I KU                          09    I S1      11    I KLU
                  ..............                        **************  ..............
                  :JOBCODE     :K                       *DED_CODE    ** :COURSE_CODE :
                  :JOB_DESC    :                        *DED_AMT     ** :COURSE_NAME :
                  :            :                        *            ** :            :
                  :            :                        *            ** :            :
                  :            :                        *            ** :            :
                  :............:                        *************** :............:
                         I JOBFILE                       ***************          EDUCFILE
                         I
```

The JOBFILE Database

The JOBFILE database contains information on a company’s job positions. Its segments are:

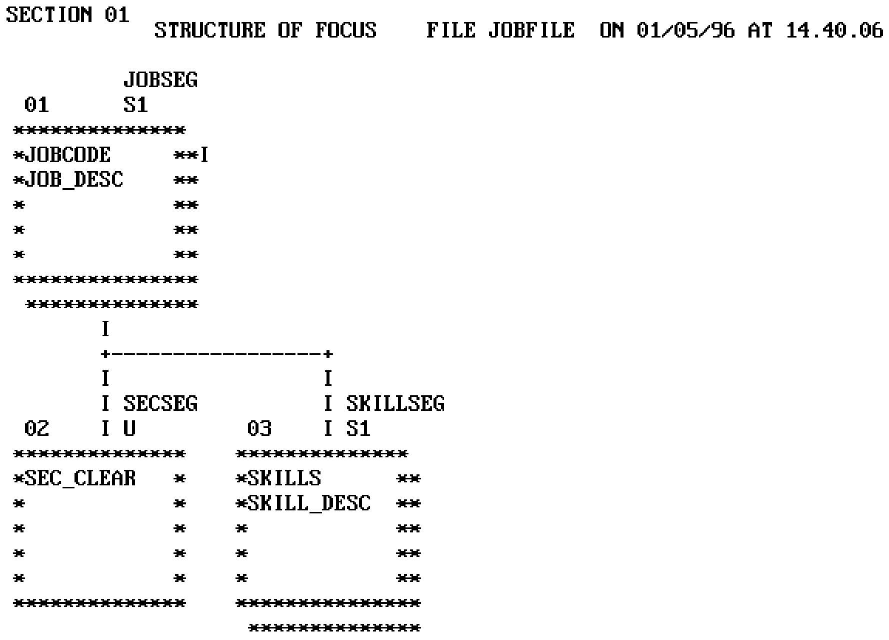
- JOBSEG describes what each position is. The field JOBCODE in this segment is indexed.
- SKILLSEG lists the skills required by each position.
- SECSEG specifies the security clearance needed, if any. This segment is unique.

The JOBFILE Master File



```
FILENAME=JOBFILE  ,SUFFIX=FOC
SEGNAME=JOBSEG   ,SEGTYPE=S1
 FIELD=JOBCODE     ,ALIAS=JC          ,USAGE=A3          ,INDEX=I,$
 FIELD=JOB_DESC    ,ALIAS=JD          ,USAGE=A25                 ,$
SEGNAME=SKILLSEG ,SEGTYPE=S1   ,PARENT=JOBSEG
 FIELD=SKILLS      ,ALIAS=            ,USAGE=A4                  ,$
 FIELD=SKILL_DESC  ,ALIAS=SD          ,USAGE=A30                 ,$
SEGNAME=SECSEG   ,SEGTYPE=U    ,PARENT=JOBSEG
 FIELD=SEC_CLEAR   ,ALIAS=SC          ,USAGE=A6                  ,$
```

The JOBFILE Structure Diagram



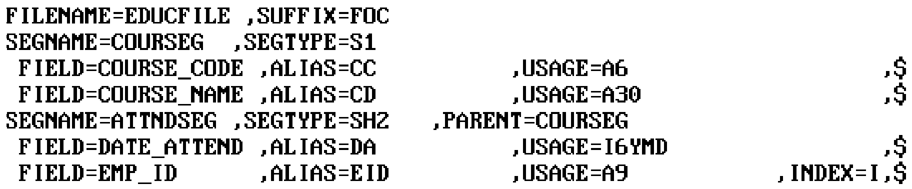
```
SECTION 01
            STRUCTURE OF FOCUS    FILE JOBFILE  ON 01/05/96 AT 14.40.06

         JOBSEG
 01      S1
**************
*JOBCODE     **I
*JOB_DESC    **
*            **
*            **
*            **
***************
 **************
       I
       +-----------------+
       I                 I
       I SECSEG          I SKILLSEG
 02    I U        03     I S1
**************    **************
*SEC_CLEAR   *    *SKILLS      **
*            *    *SKILL_DESC  **
*            *    *            **
*            *    *            **
*            *    *            **
**************    ***************
                   **************
```

The EDUCFILE Database

The EDUCFILE database contains data on a company's in-house courses. Its segments are:

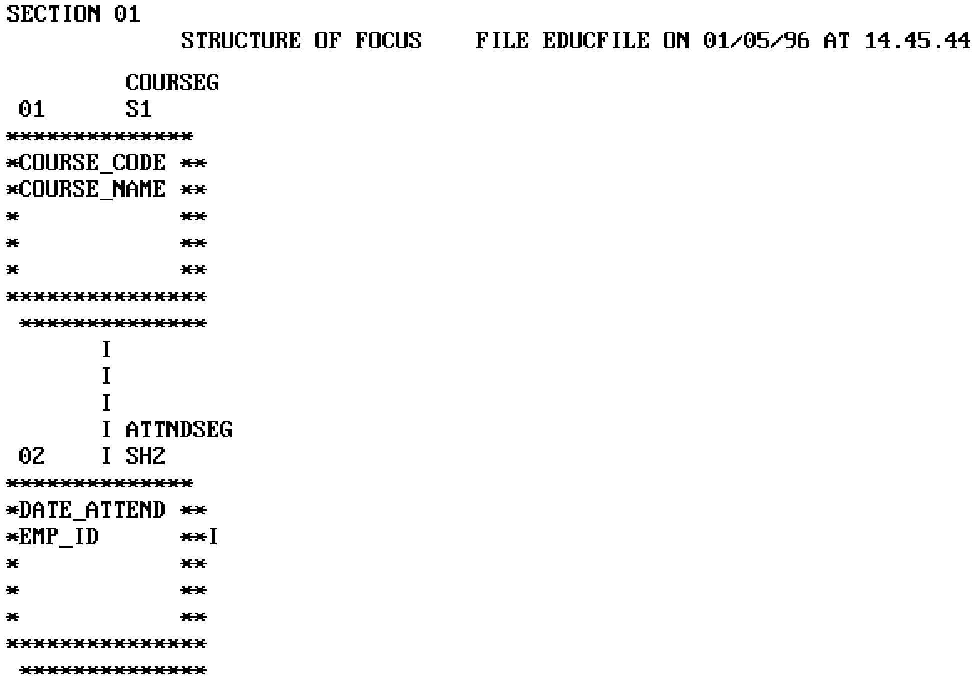
- COURSEG contains data on each course.
- ATTNDSEG specifies which employees attended the courses. Both fields in the segment are key fields. The field EMP_ID in this segment is indexed.

The EDUCFILE Master File



```
FILENAME=EDUCFILE ,SUFFIX=FOC
SEGNAME=COURSEG  ,SEGTYPE=S1
 FIELD=COURSE_CODE ,ALIAS=CC          ,USAGE=A6                    ,$
 FIELD=COURSE_NAME ,ALIAS=CD          ,USAGE=A30                   ,$
SEGNAME=ATTNDSEG ,SEGTYPE=SH2   ,PARENT=COURSEG
 FIELD=DATE_ATTEND ,ALIAS=DA          ,USAGE=I6YMD                 ,$
 FIELD=EMP_ID      ,ALIAS=EID         ,USAGE=A9            ,INDEX=I,$
```

The EDUCFILE Structure Diagram



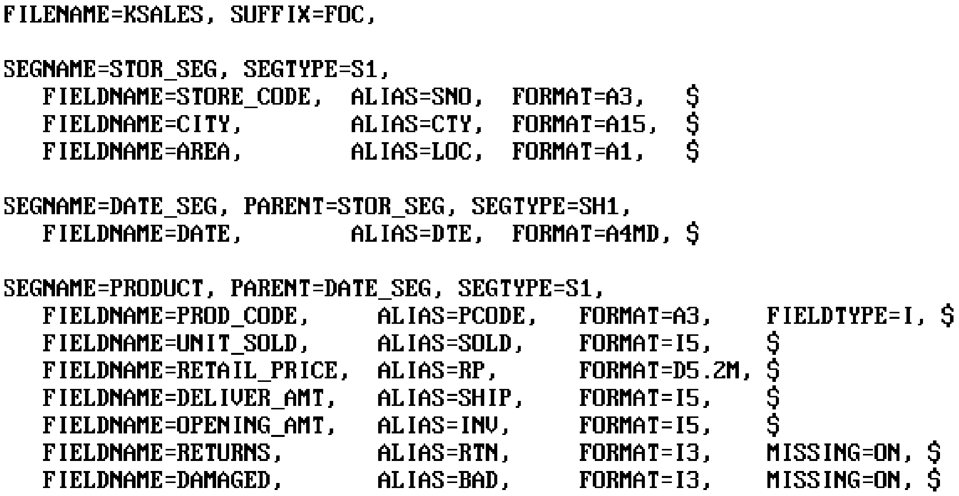
```
SECTION 01
             STRUCTURE OF FOCUS    FILE EDUCFILE ON 01/05/96 AT 14.45.44

         COURSEG
 01      S1
**************
*COURSE_CODE **
*COURSE_NAME **
*             **
*             **
*             **
***************
 **************
       I
       I
       I
       I ATTNDSEG
 02    I SH2
**************
*DATE_ATTEND **
*EMP_ID      **I
*             **
*             **
*             **
***************
 **************
```

The SALES Database

The SALES database records sales data for a dairy company (or a store chain). Its segments are:

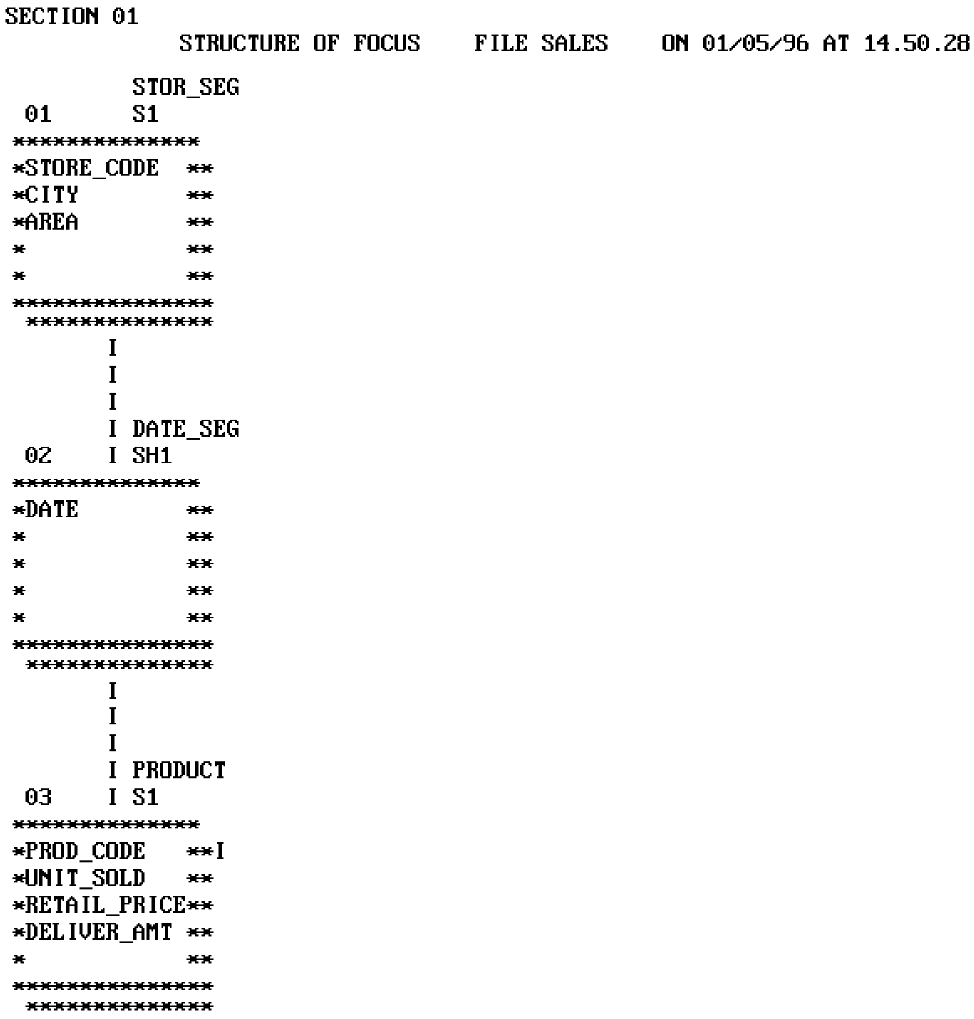
- STOR_SEG lists the stores buying the products.
- DAT_SEG contains the dates of inventory.
- PRODUCT contains sales data for each product on each date. Note the following about fields in this segment:
  - The PROD_CODE field is indexed.
  - The RETURNS and DAMAGED fields have the MISSING=ON attribute.

The SALES Master File



```
FILENAME=KSALES, SUFFIX=FOC,

SEGNAME=STOR_SEG, SEGTYPE=S1,
   FIELDNAME=STORE_CODE,  ALIAS=SNO,  FORMAT=A3,   $
   FIELDNAME=CITY,        ALIAS=CTY,  FORMAT=A15,  $
   FIELDNAME=AREA,        ALIAS=LOC,  FORMAT=A1,   $

SEGNAME=DATE_SEG, PARENT=STOR_SEG, SEGTYPE=SH1,
   FIELDNAME=DATE,        ALIAS=DTE,  FORMAT=A4MD, $

SEGNAME=PRODUCT, PARENT=DATE_SEG, SEGTYPE=S1,
   FIELDNAME=PROD_CODE,     ALIAS=PCODE,   FORMAT=A3,    FIELDTYPE=I, $
   FIELDNAME=UNIT_SOLD,     ALIAS=SOLD,    FORMAT=I5,    $
   FIELDNAME=RETAIL_PRICE,  ALIAS=RP,      FORMAT=D5.2M, $
   FIELDNAME=DELIVER_AMT,   ALIAS=SHIP,    FORMAT=I5,    $
   FIELDNAME=OPENING_AMT,   ALIAS=INV,     FORMAT=I5,    $
   FIELDNAME=RETURNS,       ALIAS=RTN,     FORMAT=I3,    MISSING=ON, $
   FIELDNAME=DAMAGED,       ALIAS=BAD,     FORMAT=I3,    MISSING=ON, $
```

The SALES Structure Diagram

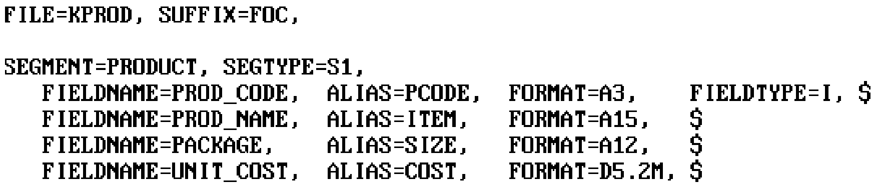


```
SECTION 01
             STRUCTURE OF FOCUS    FILE SALES    ON 01/05/96 AT 14.50.28

         STOR_SEG
 01      S1
**************
*STORE_CODE  **
*CITY        **
*AREA        **
*            **
*            **
***************
 **************
       I
       I
       I
       I DATE_SEG
 02    I SH1
**************
*DATE        **
*            **
*            **
*            **
*            **
***************
 **************
       I
       I
       I
       I PRODUCT
 03    I S1
**************
*PROD_CODE   **I
*UNIT_SOLD   **
*RETAIL_PRICE**
*DELIVER_AMT **
*            **
***************
 **************
```

The PROD Database

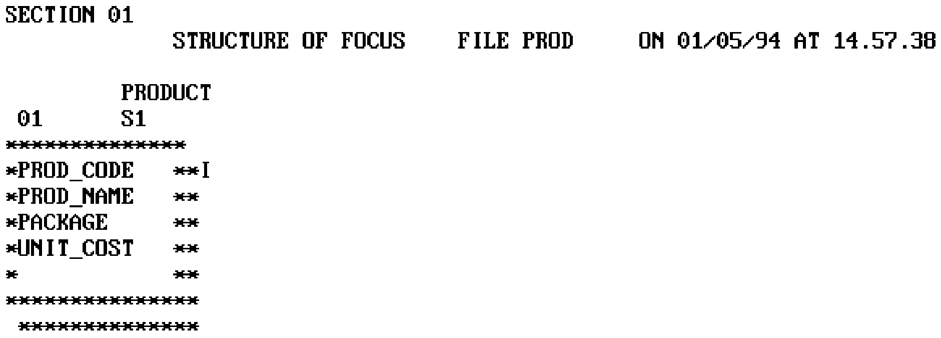
The PROD database lists products sold by a dairy company. It consists of one segment, PRODUCT. The field PROD_CODE is indexed.

The PROD Master File



```
FILE=KPROD, SUFFIX=FOC,

SEGMENT=PRODUCT, SEGTYPE=S1,
   FIELDNAME=PROD_CODE,  ALIAS=PCODE,  FORMAT=A3,    FIELDTYPE=I, $
   FIELDNAME=PROD_NAME,  ALIAS=ITEM,   FORMAT=A15,   $
   FIELDNAME=PACKAGE,    ALIAS=SIZE,   FORMAT=A12,   $
   FIELDNAME=UNIT_COST,  ALIAS=COST,   FORMAT=D5.2M, $
```

The PROD Structure Diagram



```
SECTION 01
             STRUCTURE OF FOCUS    FILE PROD     ON 01/05/94 AT 14.57.38

         PRODUCT
 01      S1
**************
*PROD_CODE   **I
*PROD_NAME   **
*PACKAGE     **
*UNIT_COST   **
*            **
***************
 **************
```

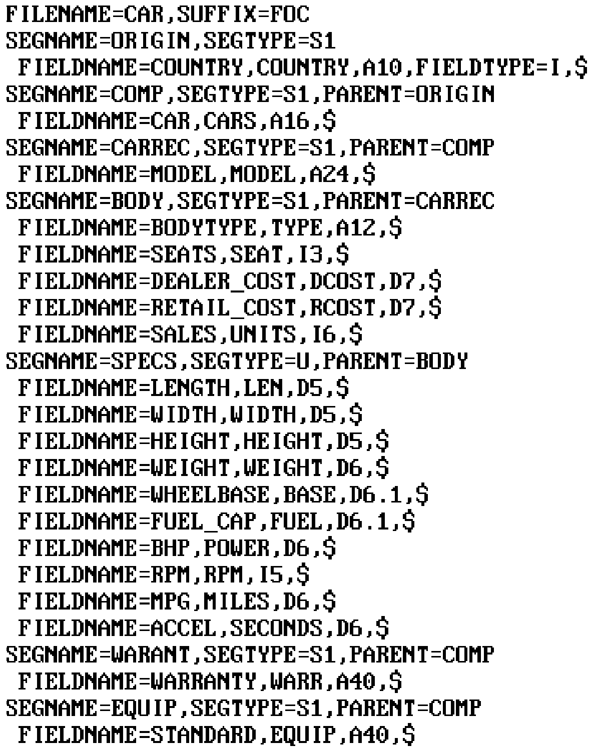
The CAR Database

The CAR database contains specifications and sales information for rare cars. Its segments are:

- ORIGIN lists the country that manufactures the car. The field COUNTRY is indexed.
- COMP contains the car name.
- CARREC contains the car model.
- BODY lists the body type, seats, dealer and retail costs, and units sold.
- SPECS lists car specifications. This segment is unique.
- WARANT lists the type of warranty.
- EQUIP lists standard equipment.

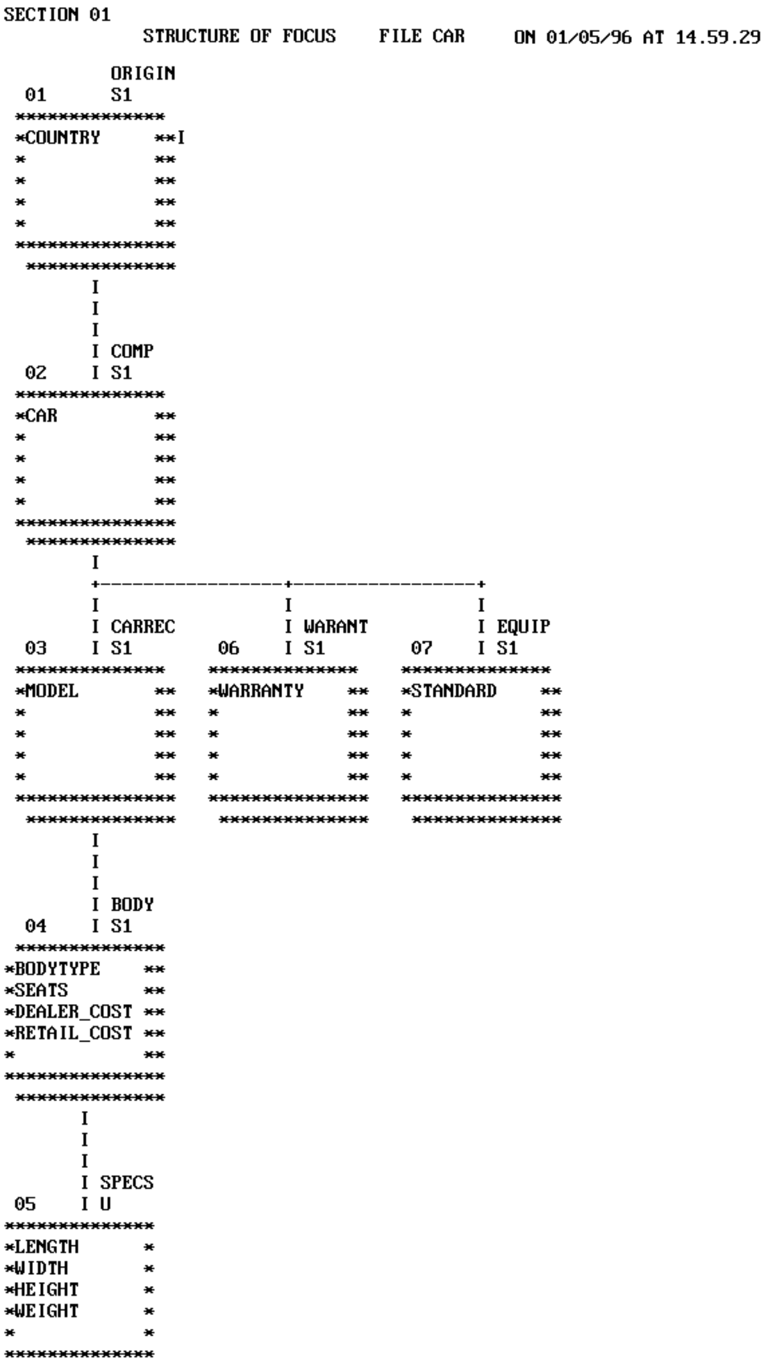
The aliases in the CAR Master File are specified without the ALIAS keyword.

The CAR Master File



```
FILENAME=CAR,SUFFIX=FOC
SEGNAME=ORIGIN,SEGTYPE=S1
 FIELDNAME=COUNTRY,COUNTRY,A10,FIELDTYPE=I,$
SEGNAME=COMP,SEGTYPE=S1,PARENT=ORIGIN
 FIELDNAME=CAR,CARS,A16,$
SEGNAME=CARREC,SEGTYPE=S1,PARENT=COMP
 FIELDNAME=MODEL,MODEL,A24,$
SEGNAME=BODY,SEGTYPE=S1,PARENT=CARREC
 FIELDNAME=BODYTYPE,TYPE,A12,$
 FIELDNAME=SEATS,SEAT,I3,$
 FIELDNAME=DEALER_COST,DCOST,D7,$
 FIELDNAME=RETAIL_COST,RCOST,D7,$
 FIELDNAME=SALES,UNITS,I6,$
SEGNAME=SPECS,SEGTYPE=U,PARENT=BODY
 FIELDNAME=LENGTH,LEN,D5,$
 FIELDNAME=WIDTH,WIDTH,D5,$
 FIELDNAME=HEIGHT,HEIGHT,D5,$
 FIELDNAME=WEIGHT,WEIGHT,D6,$
 FIELDNAME=WHEELBASE,BASE,D6.1,$
 FIELDNAME=FUEL_CAP,FUEL,D6.1,$
 FIELDNAME=BHP,POWER,D6,$
 FIELDNAME=RPM,RPM,I5,$
 FIELDNAME=MPG,MILES,D6,$
 FIELDNAME=ACCEL,SECONDS,D6,$
SEGNAME=WARANT,SEGTYPE=S1,PARENT=COMP
 FIELDNAME=WARRANTY,WARR,A40,$
SEGNAME=EQUIP,SEGTYPE=S1,PARENT=COMP
 FIELDNAME=STANDARD,EQUIP,A40,$
```

The CAR Structure Diagram

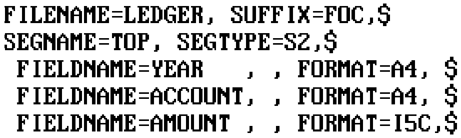


```
SECTION 01
             STRUCTURE OF FOCUS    FILE CAR     ON 01/05/96 AT 14.59.29

          ORIGIN
  01      S1
 **************
 *COUNTRY     **I
 *            **
 *            **
 *            **
 *            **
 ***************
  **************
        I
        I
        I
        I COMP
  02    I S1
 **************
 *CAR         **
 *            **
 *            **
 *            **
 *            **
 ***************
  **************
        I
        +-----------------+-----------------+
        I                 I                 I
        I CARREC          I WARANT          I EQUIP
  03    I S1       06    I S1       07    I S1
 **************    **************    **************
 *MODEL       **   *WARRANTY    **   *STANDARD    **
 *            **   *            **   *            **
 *            **   *            **   *            **
 *            **   *            **   *            **
 *            **   *            **   *            **
 ***************   ***************   ***************
  **************    **************    **************
        I
        I
        I
        I BODY
  04    I S1
 **************
*BODYTYPE    **
*SEATS       **
*DEALER_COST **
*RETAIL_COST **
*            **
***************
 **************
       I
       I
       I
       I SPECS
 05    I U
**************
*LENGTH      *
*WIDTH       *
*HEIGHT      *
*WEIGHT      *
*            *
**************
```

The LEDGER Database

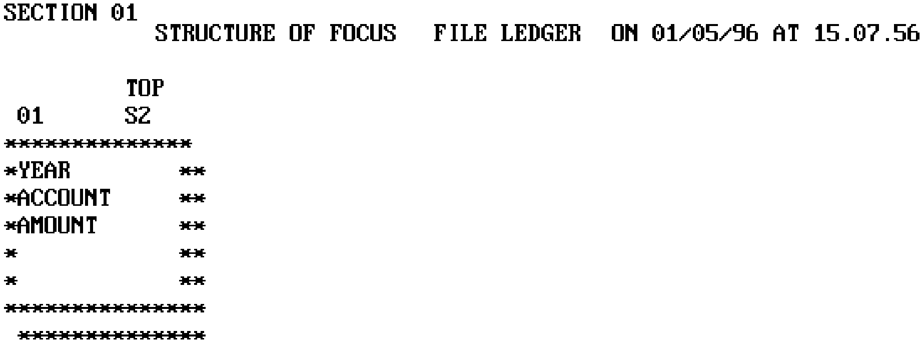
The LEDGER database lists accounting information. It consists of one segment, TOP. This database is specified primarily for EMR examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

The LEDGER Master File



```
FILENAME=LEDGER, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
 FIELDNAME=YEAR   , , FORMAT=A4, $
 FIELDNAME=ACCOUNT, , FORMAT=A4, $
 FIELDNAME=AMOUNT , , FORMAT=I5C,$
```

The LEDGER Structure Diagram

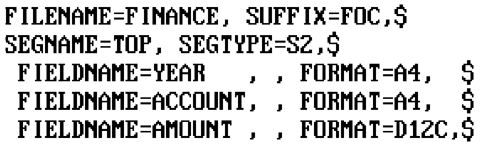


```
SECTION 01
           STRUCTURE OF FOCUS   FILE LEDGER  ON 01/05/96 AT 15.07.56

         TOP
 01      S2
**************
*YEAR        **
*ACCOUNT     **
*AMOUNT      **
*            **
*            **
***************
 **************
```

The FINANCE Database

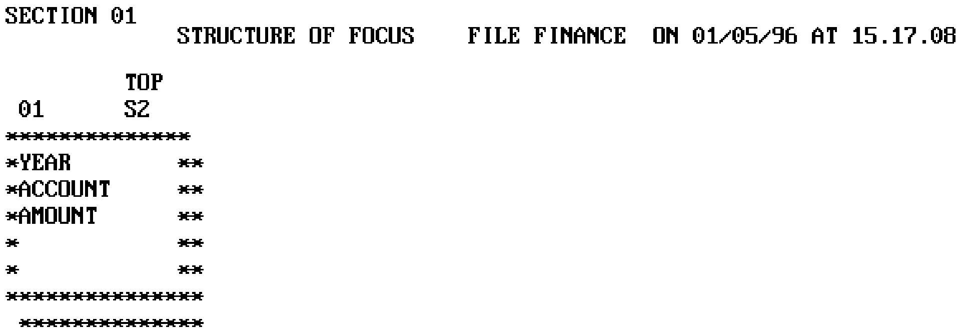
The FINANCE database contains financial information for balance sheets. It consists of one segment, TOP. This database is specified primarily for EMR examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

The FINANCE Master File



```
FILENAME=FINANCE, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
 FIELDNAME=YEAR   , , FORMAT=A4,  $
 FIELDNAME=ACCOUNT, , FORMAT=A4,  $
 FIELDNAME=AMOUNT , , FORMAT=D12C,$
```

The FINANCE Structure Diagram

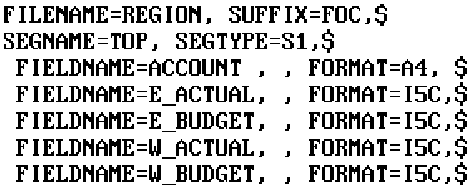


```
SECTION 01
             STRUCTURE OF FOCUS    FILE FINANCE  ON 01/05/96 AT 15.17.08

         TOP
 01      S2
**************
*YEAR        **
*ACCOUNT     **
*AMOUNT      **
*            **
*            **
***************
 **************
```

The REGION Database

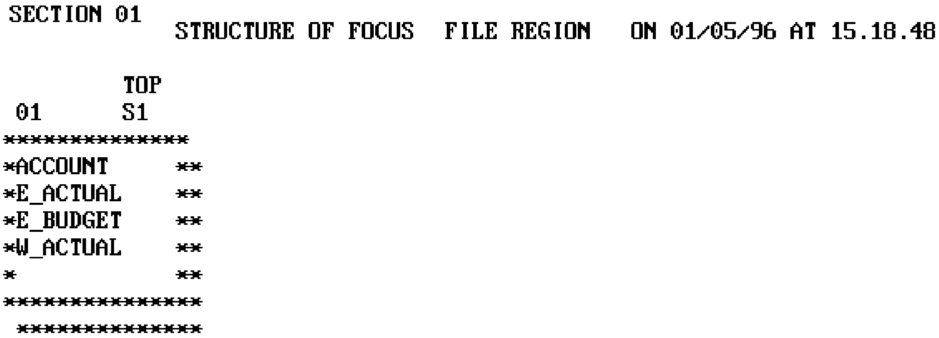
The REGION database lists account information for the east and west regions of the country. It consists of one segment, TOP. This database is specified primarily for EMR examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

The REGION Master File



```
FILENAME=REGION, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S1,$
 FIELDNAME=ACCOUNT , , FORMAT=A4, $
 FIELDNAME=E_ACTUAL, , FORMAT=I5C,$
 FIELDNAME=E_BUDGET, , FORMAT=I5C,$
 FIELDNAME=W_ACTUAL, , FORMAT=I5C,$
 FIELDNAME=W_BUDGET, , FORMAT=I5C,$
```

The REGION Structure Diagram

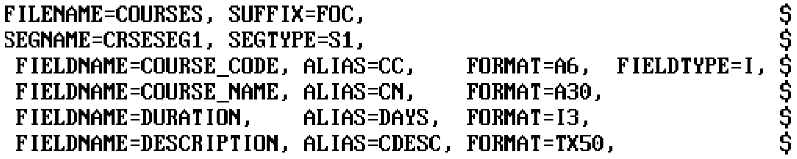


```
SECTION 01
             STRUCTURE OF FOCUS  FILE REGION   ON 01/05/96 AT 15.18.48

         TOP
 01      S1
**************
*ACCOUNT     **
*E_ACTUAL    **
*E_BUDGET    **
*W_ACTUAL    **
*            **
***************
 **************
```

The COURSES Database

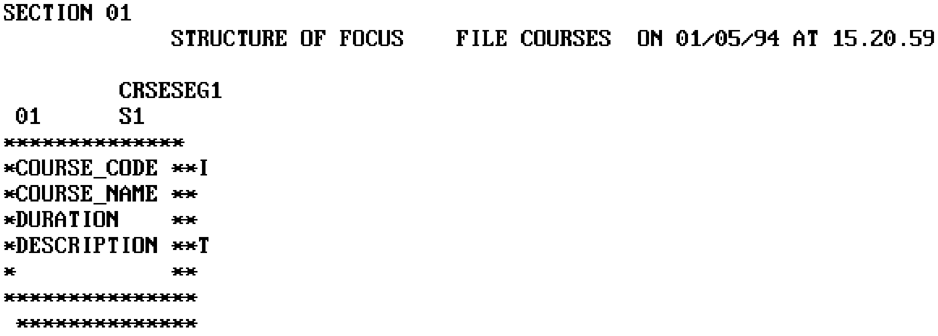
The COURSES database describes education courses. It consists of one segment, CRSESEG1. The field DESCRIPTION has a format of TEXT (TX).

The COURSES Master File



```
FILENAME=COURSES, SUFFIX=FOC,                                 $
SEGNAME=CRSESEG1, SEGTYPE=S1,                                 $
 FIELDNAME=COURSE_CODE, ALIAS=CC,    FORMAT=A6,  FIELDTYPE=I, $
 FIELDNAME=COURSE_NAME, ALIAS=CN,    FORMAT=A30,              $
 FIELDNAME=DURATION,    ALIAS=DAYS,  FORMAT=I3,               $
 FIELDNAME=DESCRIPTION, ALIAS=CDESC, FORMAT=TX50,             $
```

The COURSES Structure Diagram



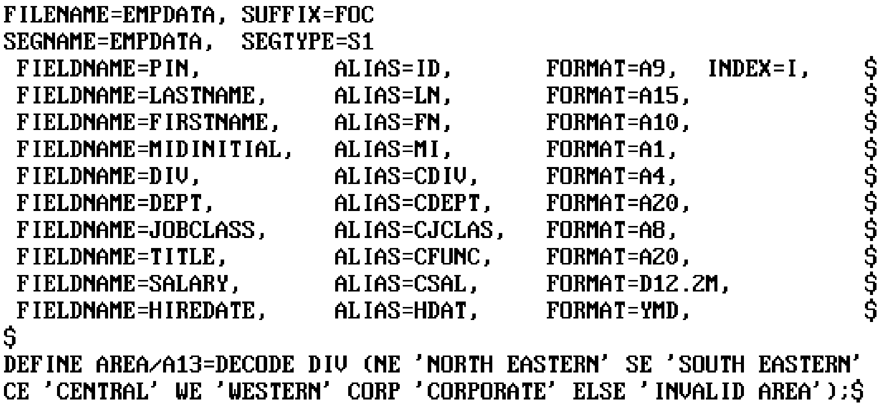
```
SECTION 01
             STRUCTURE OF FOCUS    FILE COURSES  ON 01/05/94 AT 15.20.59

         CRSESEG1
 01      S1
**************
*COURSE_CODE **I
*COURSE_NAME **
*DURATION    **
*DESCRIPTION **T
*            **
***************
 **************
```

The EMPDATA Database

The EMPDATA database contains organizational data about a company’s employees. It consists of one segment, EMPDATA. Note the following:

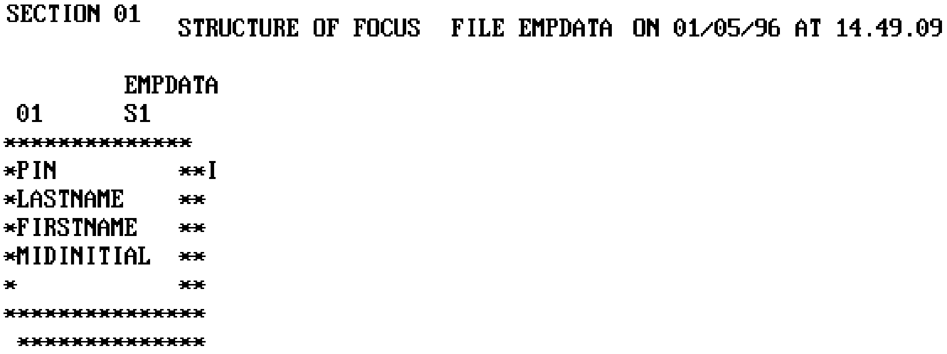
- The PIN field is indexed.
- The AREA field is a temporary one.

The EMPDATA Master File



```
FILENAME=EMPDATA, SUFFIX=FOC
SEGNAME=EMPDATA,  SEGTYPE=S1
 FIELDNAME=PIN,          ALIAS=ID,       FORMAT=A9,  INDEX=I,    $
 FIELDNAME=LASTNAME,     ALIAS=LN,       FORMAT=A15,             $
 FIELDNAME=FIRSTNAME,    ALIAS=FN,       FORMAT=A10,             $
 FIELDNAME=MIDINITIAL,   ALIAS=MI,       FORMAT=A1,              $
 FIELDNAME=DIV,          ALIAS=CDIV,     FORMAT=A4,              $
 FIELDNAME=DEPT,         ALIAS=CDEPT,    FORMAT=A20,             $
 FIELDNAME=JOBCLASS,     ALIAS=CJCLAS,   FORMAT=A8,              $
 FIELDNAME=TITLE,        ALIAS=CFUNC,    FORMAT=A20,             $
 FIELDNAME=SALARY,       ALIAS=CSAL,     FORMAT=D12.2M,          $
 FIELDNAME=HIREDATE,     ALIAS=HDAT,     FORMAT=YMD,             $
$
DEFINE AREA/A13=DECODE DIV (NE 'NORTH EASTERN' SE 'SOUTH EASTERN'
CE 'CENTRAL' WE 'WESTERN' CORP 'CORPORATE' ELSE 'INVALID AREA');$
```

The EMPDATA Structure Diagram

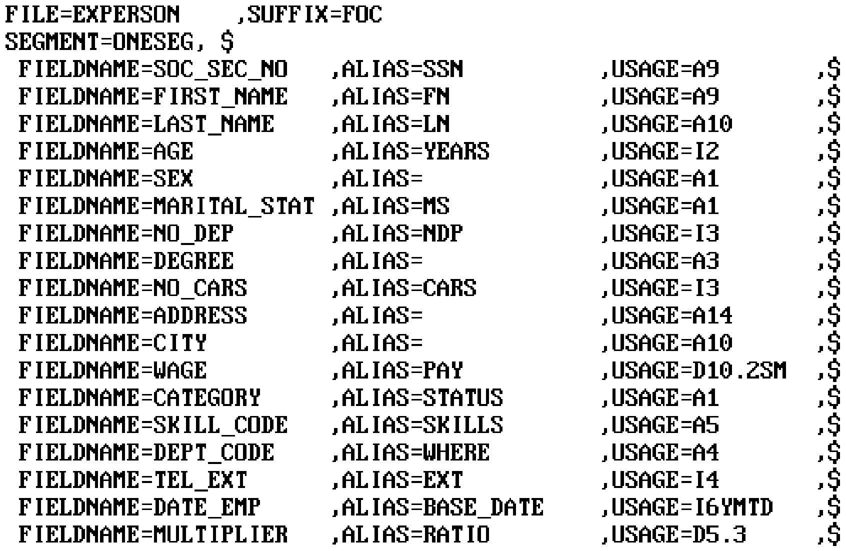


```
SECTION 01
             STRUCTURE OF FOCUS   FILE EMPDATA  ON 01/05/96 AT 14.49.09

         EMPDATA
 01      S1
***************
*PIN          **I
*LASTNAME     **
*FIRSTNAME    **
*MIDINITIAL   **
*             **
****************
 ***************
```

The EXPERSON Database

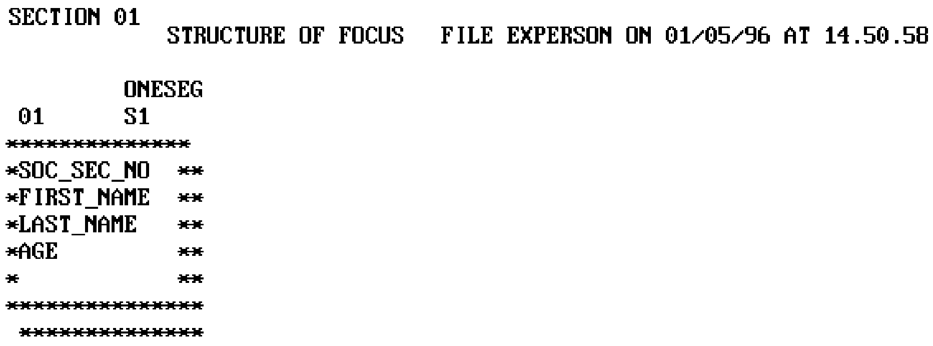
The EXPERSON database contains personal data about individual employees. It consists of one segment, ONESEG.

The EXPERSON Master File



```
FILE=EXPERSON    ,SUFFIX=FOC
SEGMENT=ONESEG, $
 FIELDNAME=SOC_SEC_NO   ,ALIAS=SSN          ,USAGE=A9        ,$
 FIELDNAME=FIRST_NAME   ,ALIAS=FN           ,USAGE=A9        ,$
 FIELDNAME=LAST_NAME    ,ALIAS=LN           ,USAGE=A10       ,$
 FIELDNAME=AGE          ,ALIAS=YEARS        ,USAGE=I2        ,$
 FIELDNAME=SEX          ,ALIAS=             ,USAGE=A1        ,$
 FIELDNAME=MARITAL_STAT ,ALIAS=MS           ,USAGE=A1        ,$
 FIELDNAME=NO_DEP       ,ALIAS=NDP          ,USAGE=I3        ,$
 FIELDNAME=DEGREE       ,ALIAS=             ,USAGE=A3        ,$
 FIELDNAME=NO_CARS      ,ALIAS=CARS         ,USAGE=I3        ,$
 FIELDNAME=ADDRESS      ,ALIAS=             ,USAGE=A14       ,$
 FIELDNAME=CITY         ,ALIAS=             ,USAGE=A10       ,$
 FIELDNAME=WAGE         ,ALIAS=PAY          ,USAGE=D10.2SM   ,$
 FIELDNAME=CATEGORY     ,ALIAS=STATUS       ,USAGE=A1        ,$
 FIELDNAME=SKILL_CODE   ,ALIAS=SKILLS       ,USAGE=A5        ,$
 FIELDNAME=DEPT_CODE    ,ALIAS=WHERE        ,USAGE=A4        ,$
 FIELDNAME=TEL_EXT      ,ALIAS=EXT          ,USAGE=I4        ,$
 FIELDNAME=DATE_EMP     ,ALIAS=BASE_DATE    ,USAGE=I6YMTD    ,$
 FIELDNAME=MULTIPLIER   ,ALIAS=RATIO        ,USAGE=D5.3      ,$
```

The EXPERSON Structure Diagram



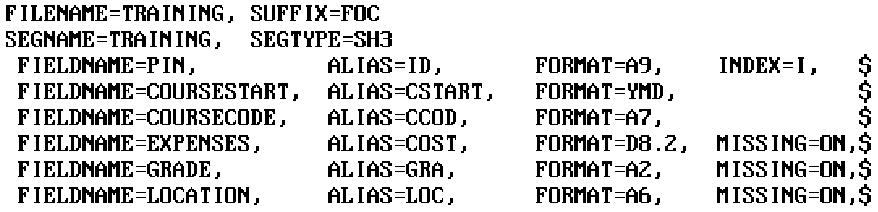
```
SECTION 01
            STRUCTURE OF FOCUS   FILE EXPERSON ON 01/05/96 AT 14.50.58

         ONESEG
 01      S1
**************
*SOC_SEC_NO  **
*FIRST_NAME  **
*LAST_NAME   **
*AGE         **
*            **
***************
 **************
```

The TRAINING Database

The TRAINING database contains training course data for employees. It consists of one segment, TRAINING. Note the following:

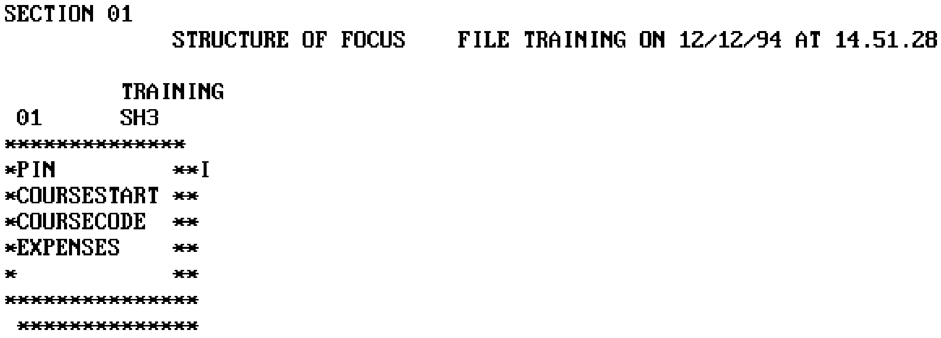
- The PIN field is indexed.
- The EXPENSES, GRADE, and LOCATION fields have the MISSING=ON attribute.

The TRAINING Master File



```
FILENAME=TRAINING, SUFFIX=FOC
SEGNAME=TRAINING,  SEGTYPE=SH3
 FIELDNAME=PIN,          ALIAS=ID,       FORMAT=A9,    INDEX=I,   $
 FIELDNAME=COURSESTART,  ALIAS=CSTART,   FORMAT=YMD,              $
 FIELDNAME=COURSECODE,   ALIAS=CCOD,     FORMAT=A7,               $
 FIELDNAME=EXPENSES,     ALIAS=COST,     FORMAT=D8.2,  MISSING=ON,$
 FIELDNAME=GRADE,        ALIAS=GRA,      FORMAT=A2,    MISSING=ON,$
 FIELDNAME=LOCATION,     ALIAS=LOC,      FORMAT=A6,    MISSING=ON,$
```

The TRAINING Structure Diagram

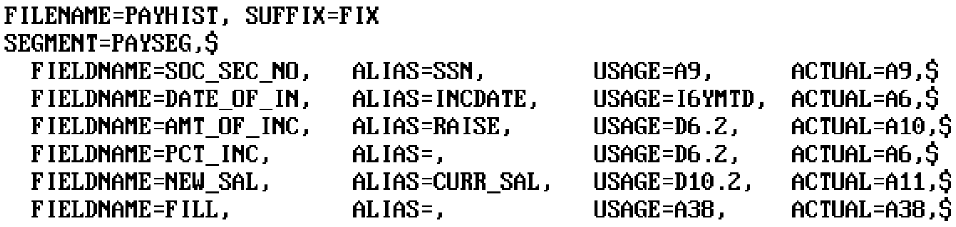


```
SECTION 01
             STRUCTURE OF FOCUS    FILE TRAINING ON 12/12/94 AT 14.51.28

         TRAINING
 01      SH3
**************
*PIN         **I
*COURSESTART **
*COURSECODE  **
*EXPENSES    **
*            **
***************
 **************
```

The PAYHIST File

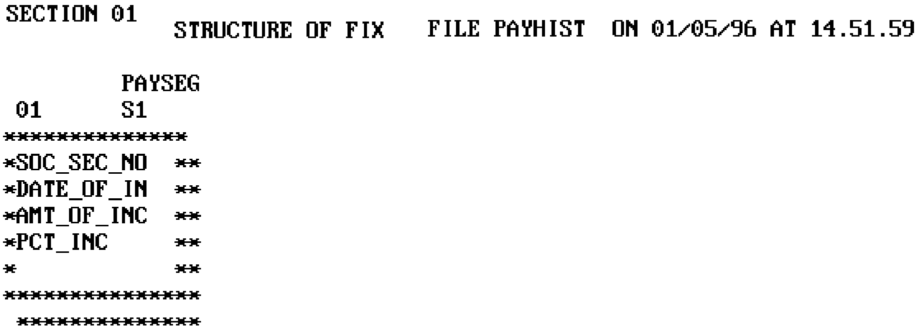
The PAYHIST database contains the employees’ salary history. It consists of one segment, PAYSEG. The SUFFIX attribute indicates that the data file is a fixed-format sequential file.

The PAYHIST Master File



```
FILENAME=PAYHIST, SUFFIX=FIX
SEGMENT=PAYSEG,$
  FIELDNAME=SOC_SEC_NO,   ALIAS=SSN,        USAGE=A9,      ACTUAL=A9,$
  FIELDNAME=DATE_OF_IN,   ALIAS=INCDATE,    USAGE=I6YMTD,  ACTUAL=A6,$
  FIELDNAME=AMT_OF_INC,   ALIAS=RAISE,      USAGE=D6.2,    ACTUAL=A10,$
  FIELDNAME=PCT_INC,      ALIAS=,           USAGE=D6.2,    ACTUAL=A6,$
  FIELDNAME=NEW_SAL,      ALIAS=CURR_SAL,   USAGE=D10.2,   ACTUAL=A11,$
  FIELDNAME=FILL,         ALIAS=,           USAGE=A38,     ACTUAL=A38,$
```

The PAYHIST Structure Diagram



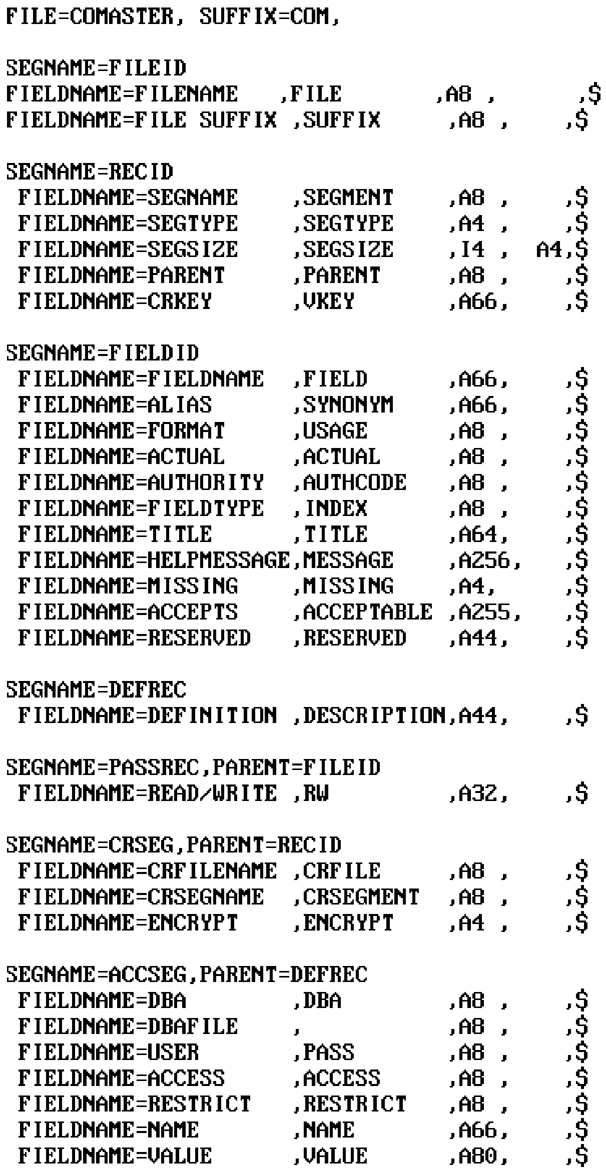
```
SECTION 01
             STRUCTURE OF FIX   FILE PAYHIST  ON 01/05/96 AT 14.51.59

         PAYSEG
 01      S1
***************
*SOC_SEC_NO  **
*DATE_OF_IN  **
*AMT_OF_INC  **
*PCT_INC     **
*            **
****************
 ***************
```

The COMASTER File

The COMASTER file is used to display the file structure and contents of each segment in a data source. Since COMASTER is used for debugging other Master Files, a corresponding FOCEXEC does not exist for the COMASTER file. Its segments are:

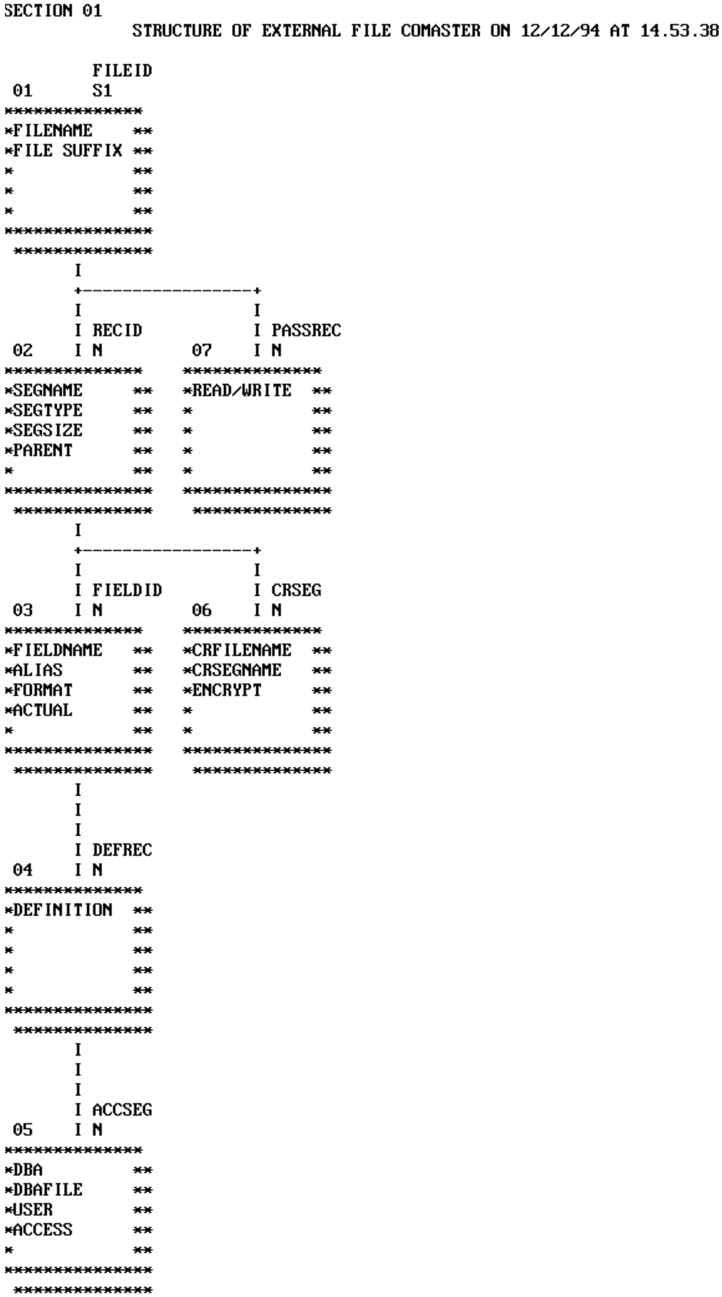
- FILEID lists file information.
- RECID lists segment information.
- FIELDID lists field information.
- DEFREC lists a description record.
- PASSREC lists read/write access.
- CRSEG lists cross-reference information for segments.
- ACCSEG lists DBA information.

The COMASTER Master File



```
FILE=COMASTER, SUFFIX=COM,

SEGNAME=FILEID
FIELDNAME=FILENAME   ,FILE       ,A8 ,      ,$
FIELDNAME=FILE SUFFIX ,SUFFIX     ,A8 ,    ,$

SEGNAME=RECID
 FIELDNAME=SEGNAME    ,SEGMENT    ,A8 ,    ,$
 FIELDNAME=SEGTYPE    ,SEGTYPE    ,A4 ,    ,$
 FIELDNAME=SEGSIZE    ,SEGSIZE    ,I4 ,  A4,$
 FIELDNAME=PARENT     ,PARENT     ,A8 ,    ,$
 FIELDNAME=CRKEY      ,VKEY       ,A66,    ,$

SEGNAME=FIELDID
 FIELDNAME=FIELDNAME  ,FIELD      ,A66,    ,$
 FIELDNAME=ALIAS      ,SYNONYM    ,A66,    ,$
 FIELDNAME=FORMAT     ,USAGE      ,A8 ,    ,$
 FIELDNAME=ACTUAL     ,ACTUAL     ,A8 ,    ,$
 FIELDNAME=AUTHORITY  ,AUTHCODE   ,A8 ,    ,$
 FIELDNAME=FIELDTYPE  ,INDEX      ,A8 ,    ,$
 FIELDNAME=TITLE      ,TITLE      ,A64,    ,$
 FIELDNAME=HELPMESSAGE,MESSAGE    ,A256,   ,$
 FIELDNAME=MISSING    ,MISSING    ,A4,     ,$
 FIELDNAME=ACCEPTS    ,ACCEPTABLE ,A255,   ,$
 FIELDNAME=RESERVED   ,RESERVED   ,A44,    ,$

SEGNAME=DEFREC
 FIELDNAME=DEFINITION ,DESCRIPTION,A44,    ,$

SEGNAME=PASSREC,PARENT=FILEID
 FIELDNAME=READ/WRITE ,RW         ,A32,    ,$

SEGNAME=CRSEG,PARENT=RECID
 FIELDNAME=CRFILENAME ,CRFILE     ,A8 ,    ,$
 FIELDNAME=CRSEGNAME  ,CRSEGMENT  ,A8 ,    ,$
 FIELDNAME=ENCRYPT    ,ENCRYPT    ,A4 ,    ,$

SEGNAME=ACCSEG,PARENT=DEFREC
 FIELDNAME=DBA        ,DBA        ,A8 ,    ,$
 FIELDNAME=DBAFILE    ,           ,A8 ,    ,$
 FIELDNAME=USER       ,PASS       ,A8 ,    ,$
 FIELDNAME=ACCESS     ,ACCESS     ,A8 ,    ,$
 FIELDNAME=RESTRICT   ,RESTRICT   ,A8 ,    ,$
 FIELDNAME=NAME       ,NAME       ,A66,    ,$
 FIELDNAME=VALUE      ,VALUE      ,A80,    ,$
```

The COMASTER Structure Diagram



```
SECTION 01
             STRUCTURE OF EXTERNAL FILE COMASTER ON 12/12/94 AT 14.53.38

         FILEID
 01      S1
**************
*FILENAME    **
*FILE SUFFIX **
*            **
*            **
*            **
***************
 **************
       I
       +-----------------+
       I                 I
       I RECID           I PASSREC
 02    I N        07     I N
**************    **************
*SEGNAME     **   *READ/WRITE  **
*SEGTYPE     **   *            **
*SEGSIZE     **   *            **
*PARENT      **   *            **
*            **   *            **
***************   ***************
 **************    **************
       I
       +-----------------+
       I                 I
       I FIELDID         I CRSEG
 03    I N        06     I N
**************    **************
*FIELDNAME   **   *CRFILENAME  **
*ALIAS       **   *CRSEGNAME   **
*FORMAT      **   *ENCRYPT     **
*ACTUAL      **   *            **
*            **   *            **
***************   ***************
 **************    **************
       I
       I
       I
       I DEFREC
 04    I N
**************
*DEFINITION  **
*            **
*            **
*            **
*            **
***************
 **************
       I
       I
       I
       I ACCSEG
 05    I N
**************
*DBA         **
*DBAFILE     **
*USER        **
*ACCESS      **
*            **
***************
 **************
```

VideoTrk and Movies Databases

The VideoTrk database tracks customer, rental, and purchase information for a video rental business. It is joined to the Movies database. VideoTrk and Movies are used in examples that illustrate the use of the Maintain facility.

VideoTrk Master

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

Movies Master

```
FILENAME=MOVIES,    SUFFIX=FOC
SEGNAME=MOVINFO,    SEGTYPE=S1
  FIELDNAME=MOVIECODE,   ALIAS=MCOD,  FORMAT=A6, INDEX=I, $
  FIELDNAME=TITLE,       ALIAS=MTL,   FORMAT=A39, $
  FIELDNAME=CATEGORY,    ALIAS=CLASS, FORMAT=A8, $
  FIELDNAME=DIRECTOR,    ALIAS=DIR,   FORMAT=A17, $
  FIELDNAME=RATING,      ALIAS=RTG,   FORMAT=A4, $
  FIELDNAME=RELDATE,     ALIAS=RDAT,  FORMAT=YMD, $
  FIELDNAME=WHOLESALEPR, ALIAS=WPRC,  FORMAT=F6.2, $
  FIELDNAME=LISTPR,      ALIAS=LPRC,  FORMAT=F6.2, $
  FIELDNAME=COPIES,      ALIAS=NOC,   FORMAT=I3, $
```

VideoTrk Structure Diagram

```
SECTION 01
             STRUCTURE OF FOCUS    FILE VIDEOTRK ON 05/21/99 AT 12.25.19

         CUST
 01      S1
**************
*CUSTID      **
*LASTNAME    **
*FIRSTNAME   **
*EXPDATE     **
*            **
***************
 **************
       I
       I
       I
       I TRANSDAT
 02    I SH1
**************
*TRANSDATE   **
*            **
*            **
*            **
*            **
***************
 **************
       I
       +-----------------+
       I                 I
       I SALES           I RENTALS
 03    I S2        04    I S2
**************    **************
*PRODCODE    **   *MOVIECODE   **I
*TRANSCODE   **   *COPY        **
*QUANTITY    **   *RETURNDATE  **
*TRANSTOT    **   *FEE         **
*            **   *            **
***************   ***************
 **************    **************
```

Movies Structure Diagram

```
SECTION 01
             STRUCTURE OF FOCUS    FILE MOVIES   ON 05/21/99 AT 12.26.05

         MOVINFO
 01      S1
**************
*MOVIECODE   **I
*TITLE       **
*CATEGORY    **
*DIRECTOR    **
*            **
***************
 **************
```

B Error Messages

If you need to see the text or explanation for any error message, you can display it online in your FOCUS session or find it in a standard FOCUS ERRORS file. All of the FOCUS error messages are stored in eight system ERRORS files:

- For CMS, these files are:

  FOT004 ERRORS

  FOG004 ERRORS

  FOM004 ERRORS

  FOS004 ERRORS

  FOA004 ERRORS

  FSQLXLT ERRORS

  FOCSTY ERRORS

  FOB004 ERRORS

- For MVS, these files are the following members in the ERRORS PDS:

  FOT004

  FOG004

  FOM004

  FOS004

  FOA004

  FSQLXLT

  FOCSTY

  FOB004

To display a message online, issue the following query command at the FOCUS command level

```
? n
```

where *n* is the message number.

The message number and text will display along with a detailed explanation of the message (if one exists). For example, issuing the following command:

```
? 210
```

displays the following

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

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

C Syntax Summary

Topics:

This chapter describes FOCUS reporting commands and options.

TABLE Syntax Summary

The syntax of a TABLE request is:

```
DEFINE FILE filename  [CLEAR]
                      [ADD  ]
tempfield[/format]      {[WITH realfield] = expression;             }
                        {REDEFINES qualifier.fieldname=expression;  }
.
.
.
END

TABLE FILE filename
HEADING [CENTER]
"text"
{PRINT}{[SEG.]field     [/R]   [/format]                        }
{LIST }{                [/L]                                    }
{SUM  }{                [/C]                                    }
{WRITE}{ [prefixoperator.][field]    [/R]   [/format]           }
{ADD  }     *                        [/L]
{COUNT}                              [/C]
    [NOPRINT                ]     [AND ]    obj2...obj256
    [AS 'title1,...,title5' ]     [OVER]
     [WITHIN field]
```

```
COMPUTE field[/format ]=expression;[AS 'title,...,title5'] [IN n]

[AND] ROW-TOTAL       ⎡/R⎤ [/format][AS 'name']
                      ⎢/L⎥
                      ⎣/C⎦

[AND] COLUMN-TOTAL    ⎡/R⎤   [AS 'name']
                      ⎢/L⎥
                      ⎣/C⎦

ACROSS [HIGHEST] sortfieldn [IN-GROUPS-OF qty]     ⎡NOPRINT               ⎤
                                                   ⎣AS'title1,...,title5' ⎦
BY [HIGHEST] sortfieldn [IN-GROUPS-OF qty]         ⎡NOPRINT               ⎤
                                                   ⎣AS'title1,...,title5' ⎦

RANKED BY  ⎧TOP    ⎫   [n]field
           ⎨HIGHEST⎬
           ⎩LOWEST ⎭

ON sortfield option1 [AND] option2 [WHEN expression;...]

ON sortfield PCSEND LOCATION /dir AS burstname] FORMAT HTML

WHERE  ⎧[TOTAL] expression⎫
       ⎨RECORDLIMIT EQ n  ⎬
       ⎩READLIMIT EQ n    ⎭

IF [TOTAL] field relation value [OR value...]

ON TABLE SET parameter value

ON TABLE   ⎧HOLD [VIA program]⎫
           ⎪PCHOLD            ⎪
           ⎨SAVE              ⎬[AS name] [FORMAT format]⎡MISSING  ⎧ON ⎫ ⎤
           ⎩SAVB              ⎭                          ⎣         ⎨OFF⎬ ⎦
                                                                   ⎩   ⎭

ON TABLE   ⎧NOTOTAL                                                ⎫
           ⎨COLUMN-TOTAL     ⎡/R⎤  [AS 'name']  fieldname          ⎬
           ⎪                 ⎢/L⎥                                  ⎪
           ⎪                 ⎣/C⎦                                  ⎪
           ⎪                                                       ⎪
           ⎪ROW-TOTAL        ⎡/R⎤  [format] [AS 'name']  fieldname ⎪
           ⎪                 ⎢/L⎥                                  ⎪
           ⎩                 ⎣/C⎦                                  ⎭

FOOTING [CENTER] [BOTTOM]
"text"
MORE
FILE file2

   ⎡IF field relation value [OR value...]⎤
   ⎣WHERE expression                     ⎦
⎧END ⎫
⎨RUN ⎬
⎩QUIT⎭
```

TABLEF Syntax Summary

The syntax of a TABLEF request is:

```
TABLEF FILE filename
HEADING [CENTER]
"text"
{PRINT}    [SEG.]field               [NOPRINT                      ] [IN n]
{LIST }    [prefixoperator.]field    [AS 'title1,title2,...,title5']
{SUM  }
{WRITE}
{COUNT}   [AND ]     field2 . . . field256
          [OVER]

COMPUTE field [/format]=expression; [AS 'title1,...title5']
[AND] ROW-TOTAL [AND] COLUMN-TOTAL

BY [HIGHEST] keyfieldn [NOPRINT]

ON keyfield option1 [AND] option2. . . .

WHERE [TOTAL] expression

IF [TOTAL] field relation value [OR value. . . ]

ON TABLE SET parameter value

ON TABLE   {HOLD [VIA program]}  [AS name] [FORMAT format]  [MISSING {ON }]
           {PCHOLD            }                             [        {OFF}]
           {SAVE              }                             [             ]
           {SAVB              }

ON TABLE  {NOTOTAL                     }
          {COLUMN-TOTAL fieldname      }
          {ROW-TOTAL fieldname         }

 FOOTING [CENTER] [BOTTOM]
 "text"

{END }
{RUN }
{QUIT}
```

Note:

- Prefix operators for TABLE can be: AVE., ASQ., MAX., MIN., PCT., RPCT., PCT.CNT., FST., LST., CNT., SUM., or TOT. TABLEF requests cannot use prefix operators PCT.CNT., RPCT., and TOT.
- For HOLD files, formats can be: ALPHA, DIF, LOTUS, SYLK, WP, SQL, DB2, SQLDBC, FOCUS, HTML, or HTMTABLE; for PCHOLD files, formats can be: DIF, LOTUS, SYLK, WP, HTML, or HTMTABLE; for SAVE files, formats can be: DIF, LOTUS, SYLK, or WP.
- In a BY phrase, you may have up to 32 fields.

MATCH Syntax Summary

The syntax of a MATCH request is:

```
MATCH FILE filename     (the OLD file)

report request

BY field1 [AS sortfield]

MORE
FILE file3
subrequest

RUN
.
.
.
FILE filename2          (the NEW file)

report request

BY field1 [AS sortfield1]
.
.
.
[AFTER MATCH HOLD [AS filename] [FORMAT FOCUS]      {OLD          }]
[                                                   {NEW          }]
[                                                   {OLD-NOT NEW  }]
[                                                   {NEW-NOT-OLD  }]
[                                                   {OLD-AND-NEW  }]
[                                                   {OLD-OR-NEW   }]
[                                                   {OLD-NOR-NEW  }]

MORE
FILE file4
subrequest

END
```

FOR Syntax Summary

The formal syntax of the FOR statement is

```
FOR fieldname [NOPRINT]
line
[OVER line]
.
.
.
.
END
```

where:

line

Can be any of the following:

```
tag [OR tag...][options]
[fieldname]
DATA n,[n,....] $
DATA PICKUP [FROM filename] tag [LABEL label] [AS 'text']
RECAP name[/format]=expression;
BAR [AS 'character'] [OVER]
"text"
PAGE-BREAK [OVER]
```

tag

Can be any of the following:

```
value [OR value...]
value TO value
```

option

Can be any of the following:

```
AS 'text'
NOPRINT

[LABEL label]

WHEN EXISTS
NOPRINT

[POST [TO filename]]
```

D Creating Your Own Subroutines

Topics:

- Process Overview
- Considerations for Writing Subroutines
- Compilation and Storage
- Testing the Subroutine
- Example of a Custom Subroutine: The MTHNAM Subroutine

This appendix discusses how to create your own private collection of subroutines to use with FOCUS.

Process Overview

The process of creating a subroutine involves four steps:

1. Write the subroutine for FOCUS the same way you would for a program. Use any language that supports subroutine calls; among the most common languages are FORTRAN, COBOL, PL/I, Assembler, and C.
2. Store the subroutine in a separate file; do not include it in the main program.
3. Compile the subroutine. In MVS, link-edit it; in CMS, add the subroutine to a load library using the GENSUBLL command.
4. Test the subroutine; specify it in a FOCUS command, report request, or procedure.

For example, suppose you write a program named INTCOMP that calculates the amount of money in an account earning simple interest. The program reads a record, tests if the data is acceptable, and then calls a subroutine called SIMPLE that computes the amount of money. The program and the subroutine are stored together in the same file.

The program and the subroutine shown here are written in pseudocode (a method of representing computer code in a general way):

```
Begin program INTCOMP.
Execute this loop until end-of-file.
    Read next record, fields: PRINCPAL, DATE_PUT, YRRATE.
    If PRINCPAL is negative or greater than 100,000,
        reject record.
    If DATE_PUT is before January 1, 1975, reject record.
    If YRRATE is negative or greater than 20%, reject record.
    Call subroutine SIMPLE (PRINCPAL, DATE_PUT, YRRATE, TOTAL).
    Print PRINCPAL, YEARRATE, TOTAL.
End of loop.
End of program.

Subroutine SIMPLE (AMOUNT, DATE, RATE, RESULT).
Retrieve today's date from the system.
Let NO_DAYS = Days from DATE until today's date.
Let DAY_RATE = RATE / 365 days in a year.
Let RESULT = AMOUNT * (NO_DAYS * DAY_RATE + 1).
End of subroutine.
```

If you move the SIMPLE subroutine into a file separate from the main program and compile it, you can call the subroutine from FOCUS. The following report request shows how much money employees would accrue if they invested their salaries in accounts paying 12%:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME DAT_INC SALARY AND COMPUTE
    INVESTED/D10.2 = SIMPLE (SALARY, DAT_INC, 0.12, INVESTED);
BY EMP_ID
END
```

Note: The subroutine is designed to return only the amount of the investment, not today’s date. This is because a subroutine can return only a single value to FOCUS each time it is called.

Considerations for Writing Subroutines

When you write a subroutine for FOCUS, there are requirements and limits which you need to consider. The section provides information about:

- Naming conventions
- Argument considerations
- Programming considerations
- Language considerations
- A programming technique that uses entry points. Entry points enable you to use one algorithm to produce different results.
- A programming technique that allows multiple subroutine calls. Multiple calls enable the subroutine to process more than 28 arguments.

Naming Conventions

The subroutine name may consist of up to eight characters, unless the language you are using to write the subroutine supports a shorter naming convention. Each character can be a letter or number. The first character of the name must be a letter (A-Z). Special symbols are not permitted.

Argument Considerations

When you create your arguments, consider these points:

- **The argument maximum.** Subroutine calls in FOCUS may contain up to 28 arguments. However, you can bypass this restriction if you create a subroutine that accepts multiple calls, as described in *Programming Technique: Subroutines With More Than 28 Arguments* on page D-8.
- **Types of arguments.** Subroutine calls can serve as arguments in other subroutine calls or in FOCUS functions. For types of acceptable arguments and rules, see Chapter 9, *Using Functions and Subroutines*.
- **Input arguments.** FOCUS passes input arguments to subroutines using standard conventions. Register 1 points to the list of argument addresses. Each address is a full word.
- **Output arguments.** Subroutines may return only one output argument to the FOCUS request. Place this argument last in the subroutine argument list. You can choose any format for the output argument except in Dialogue Manager statements.

- **Internal processing.** When you specify values for arguments and FOCUS passes the arguments to a subroutine,
  - Alphanumeric arguments remain unchanged.
  - Numeric arguments are converted to 8-byte, double-precision data (except in -CMS RUN and -MVS RUN statements and amper variables, as discussed below).

    Various languages represent double-precision fields as declarations:

Language	**Declaration**
Assembler	DS, D
C	Double
COBOL	COMP-2
FORTRAN	REAL*8
PL/I	DECIMAL FLOAT (16)

- **Dialogue Manager requirements.** If you are writing a subroutine specifically for Dialogue Manager, you may need to code your subroutine to perform conversion for these situations:
  - Operating system -RUN statements. FOCUS passes all arguments from -CMS RUN, -TSO RUN, and -MVS RUN statements as alphanumeric data. If your subroutine requires numeric arguments, you may choose to have your subroutine convert these arguments into numeric format. Otherwise, the user can use the ATODBL subroutine to convert the arguments into double-precision format before passing them to the subroutine. The ATODBL subroutine is described in Chapter 9, *Using Functions and Subroutines*.
  - Operating system -RUN statements and output argument format. If the subroutine is called from a -CMS or -TSO RUN statement, the output argument is stored in the output variable in numeric format. Since FOCUS cannot interpret data stored in Dialogue Manager variables in numeric format, the data is unreadable. To prevent this, have your subroutine convert the output value into a character string.
  - -SET and output argument format. If the output argument is in numeric format, the -SET statement truncates the output value to an integer, converts it to a character string, and stores the value in a specified amper variable. To prevent this, have your subroutine convert the output value into a character string. This enables the numeric value to be passed to Dialogue Manager without being truncated to an integer.

Programming Considerations

When you plan your programming requirements, consider these points:

- Write the subroutine as a proper subroutine, not as a function.
- If the subroutine initializes variables, it must initialize them each time it is executed (serial reusability).
- Since a single FOCUS request may execute a subroutine hundreds or even thousands of times, code the subroutine as efficiently as possible.
- If you create your own subroutines in text files or text libraries, the subroutine must be 31-bit addressable.

Language Considerations

Language considerations include:

- **Available memory.**

  If you write the subroutine in a language that brings libraries into memory (for example, FORTRAN and COBOL), the libraries reduce the amount of memory available to the subroutine.

- **FORTRAN input/output operations (I/O).**

  In CMS, FOCUS does not support FORTRAN input/output operations. If a subroutine written in FORTRAN must read or write data, write the I/O portions in a separate subroutine in another language.

  In MVS/TSO, FOCUS does support FORTRAN input/output operations.

- **PL/I notes:**
  - Do not use the RETURNS attribute.
  - Include the following attribute in the procedure (PROC) statement:

    ```
    OPTIONS (COBOL)
    ```

  - Declare alphanumeric arguments received from FOCUS requests as

    ```
    CHARACTER (n)
    ```

    where *n* is the field length as defined by the FOCUS request. Do not use the VARYING attribute.

  - Declare numeric arguments received from FOCUS requests as

    `DECIMAL FLOAT (16)`

    or

    `BINARY FLOAT (53)`

  - The format of the output argument to be returned to the FOCUS request depends on how the format is described in the DEFINE or COMPUTE commands:

FOCUS Format	**PL/I Declaration**
`A`*`n`*	`CHARACTER (`*`n`*`)` (Do not use the VARYING attribute.)
`I`	`BINARY FIXED (31)`
`F`	`DECIMAL FLOAT (6)` or `BINARY FLOAT (21)`
`D`	`DECIMAL FLOAT (16)` or `BINARY FLOAT (53)`
`P`	`DECIMAL FIXED (15)` (for small packed numbers, 8 bytes) `DECIMAL FIXED (31)` (for large packed numbers, 16 bytes)

  - Declare variables that are not arguments with the STATIC attribute. This avoids dynamically allocating these variables every time the subroutine is executed.

- **C language notes:**

  - Do not return a value with the return statement.

  - Declare double-precision fields as ‘double’.

  - The format of the output parameter to be returned to the FOCUS request depends on how the format is defined in the request, as shown by the chart below:

FOCUS Format	**C Declaration**
`A`*`n`*	`char `*`*xxx n`* (**Note:** Alphabetical fields are not terminated with a null byte and, therefore, cannot be processed by many of the string manipulation subroutines in the run-time library.)
`I`	`long `*`*xxx`*
`F`	`float `*`*xxx`*
`D`	`double `*`*xxx`*
`P`	No equivalent in C.

Programming Technique: Entry Points

Normally, subroutines are executed starting from their first statement. However, they can be executed starting from any place in their code if you designate that place as an *entry point*. (How you designate entry points depends on the language you are using.) Each entry point has a name.

To execute a subroutine at an entry point, specify the entry name in the subroutine call instead of the subroutine name. The general syntax is

```
{subroutine|entrypoint}  (input1, input2,...{'format'|outfield})
```

Entry points enable a subroutine to use one basic algorithm to produce different results. For example, the DOWK subroutine calculates the days of the week on which dates fall. When you specify the subroutine name DOWK, you obtain a 3-letter abbreviation of the day. If you specify the entry name DOWKL, you obtain the full name. The calculation, however, is the same.

Entry Point Example

This example illustrates how entry points work. The FTOC subroutine, written in pseudocode below, converts Fahrenheit temperatures to Centigrade. The entry point FTOK (designated by the Entry statement) sets a flag that causes 273 to be subtracted from the Centigrade temperature (Kelvin temperature). The subroutine is:

```
Subroutine FTOC (FAREN, CENTI).
Let FLAG = 0.
Go to label X.
Entry FTOK (FAREN, CENTI).
Let FLAG = 1.
Label X.
Let CENTI = (5/9) * (FAREN - 32).
If FLAG = 1 then CENTI = CENTI - 273.
Return.
End of subroutine.
```

Here is a shorter way to write the subroutine. Notice that the *kelv* output argument listed for the entry point is different from the *centi* output argument listed at the beginning of the subroutine:

```
Subroutine FTOC (FAREN, CENTI).
Entry FTOK (FAREN, KELV).
Let CENTI = (5/9) * (FAREN - 32).
KELV = CENTI - 273.
Return.
End of Subroutine.
```

To obtain the Centigrade temperature, specify the subroutine name FTOC in the subroutine call. For example:

```
CENTIGRADE/D6.2 = FTOC (TEMPERATURE, CENTIGRADE);
```

To obtain the Kelvin temperature, specify the entry name FTOK in the subroutine call. For example:

```
KELVIN/D6.2 = FTOK (TEMPERATURE, KELVIN);
```

Note: In CMS, subroutines can be executed from their entry points only if the subroutines are stored in libraries. You must specify these libraries in the GLOBAL command, as described in *CMS: Compilation and Storage* on page D-12.

Programming Technique: Subroutines With More Than 28 Arguments

Subroutine call syntax cannot specify more than 28 arguments, including the output argument. To process more than 28 arguments, you must write the subroutine so that the user can specify two or more call statements to pass the arguments to the subroutine.

We recommend the following technique for writing subroutines with multiple call statements:

1. Divide the subroutine into segments. Each segment will receive the arguments passed by one corresponding subroutine call.

   The argument list in the beginning of your subroutine must represent the same number of arguments in the subroutine call, including a call number argument and an output argument.

   You may process some of the arguments as dummy arguments if you have an unequal number of arguments. For example, if you divide 32 arguments among six segments, the each segment processes six arguments; the sixth segment processes two arguments and four dummy arguments.

2. Include a statement at the beginning of the subroutine that reads the call number (first argument) and branches to a corresponding segment. Each segment processes the arguments from one call. (For example, number 1 branches to the first segment, number 2 to the second segment, and so on.)

3. Have each segment store the arguments it receives in other variables (which can be processed by the last segment) or accumulate them in a running total.

   End each segment with a statement returning control back to the FOCUS request (RETURN statement).

4. The last segment returns the final output value to the FOCUS request.

The following sample of pseudocode illustrates the four steps:

```
1. Subroutine name (num, input1, input2, input3, input4, outfield).
2. If NUM is 1 then goto label ONE
   else goto label TWO.

   Label ONE.
3. Let variable = input1 + input2.
   Return.

4. Label TWO
   LET outfield = variable + input3 + input4
   Return
   End of subroutine
```

Note: You can also use the entry point technique, described in *Programming Technique: Entry Points* on page D-7, to write subroutines that process more than 28 arguments.

Syntax for Subroutines With Multiple Call Statements

To use a subroutine that requires more than 28 arguments, you must specify two or more call statements to pass the arguments to the subroutine.

The syntax for calling a subroutine with multiple call statements is

```
dummy = subroutine (1, group1, dummy);
dummy = subroutine (2, group2, dummy);
    .
    .
    .
outfield = subroutine (n, groupn, outfield);
```

where:

dummy

Is either the name of a dummy field or its format, enclosed in single quotation marks. It must have the same format as the *outfield* argument.

Note: Do not specify the *dummy* argument for the last call statement; use the *outfield* argument.

subroutine

Is the name of the subroutine, up to eight characters long, depending on your programming language.

n

Is a number that identifies each subroutine call. It must be the first argument in each subroutine call. The subroutine uses this call number to branch to segments of code.

group1...

Are lists of input arguments passed by each subroutine call. Each group contains the same number of arguments, but no more than 26 arguments.

```
26 + call number + output =28
```

outfield

Is the output field that contains the value returned by the subroutine. It is the fieldname of the field that contains the output or the format of the output value, enclosed in single quotation marks, depending on the application. It is last argument in the last call.

Note:

- Each subroutine call contains the same number of arguments. This is because the argument list in each call must correspond to the argument list in the beginning of the subroutine. The last call may contain several dummy arguments.
- Subroutines may require additional arguments as determined by the programmer who created the subroutine.

Example Creating a Subroutine With 32 Input Arguments

This example illustrates how to create a subroutine with 32 input arguments using the recommended technique. It also shows how the subroutine is specified in a DEFINE command.

The ADD32 subroutine, written in pseudocode, sums 32 numbers. It is divided into six segments, each of which adds six numbers from a subroutine call. (The total number of input arguments is 36 but the last four are dummy arguments.) The sixth segment adds two arguments to the SUM variable and returns the final output value. The sixth segment does not process any values supplied for the four dummy arguments.

The subroutine is:

```
Subroutine ADD32 (NUM, A, B, C, D, E, F, TOTAL).
If NUM is 1 then goto label ONE
else if NUM is 2 then goto label TWO
else if NUM is 3 then goto label THREE
else if NUM is 4 then goto label FOUR
else if NUM is 5 then goto label FIVE
else goto label SIX.

Label ONE.
Let SUM = A + B + C + D + E + F.
Return.

Label TWO
Let SUM = SUM + A + B + C + D + E + F
Return

Label THREE
Let SUM = SUM + A + B + C + D + E + F
Return

Label FOUR
Let SUM = SUM + A + B + C + D + E + F
Return

Label FIVE
Let SUM = SUM + A + B + C + D + E + F
Return

Label SIX
LET TOTAL = SUM + A + B
Return
End of subroutine
```

To use the ADD32 subroutine, list all six call statements; each call specifying six numbers. The last four numbers, represented by zeroes, are dummy arguments. In this example, the DEFINE command stores the total of the 32 numbers in the SUM32 field.

```
DEFINE FILE EMPLOYEE
DUMMY/D10 = ADD32 (1, 5, 7, 13, 9, 4, 2, DUMMY);
DUMMY/D10 = ADD32 (2, 5, 16, 2, 9, 28, 3, DUMMY);
DUMMY/D10 = ADD32 (3, 17, 12, 8, 4, 29, 6, DUMMY);
DUMMY/D10 = ADD32 (4, 28, 3, 22, 7, 18, 1, DUMMY);
DUMMY/D10 = ADD32 (5, 8, 19, 7, 25, 15, 4, DUMMY);
SUM32/D10 = ADD32 (6, 3, 27, 0, 0, 0, 0, SUM32);
END
```

Compilation and Storage

Once you have written your subroutine, you need to compile and store it. This section discusses compiling and storing your subroutine for CMS and MVS.

CMS: Compilation and Storage

On CMS, compile the subroutine and use the GENSUBLL command to add the compiled object code to a load library (filetype LOADLIB). Enter:

```
GENSUBLL ?
```

to display for online information about the command. Do not store subroutine in the FUSELIB load library (FUSELIB LOADLIB), as it may be overwritten when your site installs the next release of FOCUS.

You may also compile the subroutine and store the compiled object code either as a text file (filetype TEXT), or as a member in a text library (filetype TXTLIB). Do not store it in the FUSELIB text library (FUSELIB TXTLIB), as it may be overwritten when your site installs the next release of FOCUS.

Individual text files are easier to maintain and control. Text libraries, on the other hand, enable you to build different entry points into the subroutine (as shown in *Programming Technique: Entry Points* on page D-7). Note that there are two CMS commands regarding text libraries:

- The TXTLIB command allows you to create, add to, and delete text libraries.
- The GLOBAL TXTLIB command allows users to specify text libraries to gain access to their subroutines.

If the subroutine is written in PL/I, append this line at the end of the text file

```
ENTRY subroutine
```

where:

subroutine

Is the name of the subroutine. You can do this using your system editor.

Make sure that any subroutines that your subroutine calls are also compiled and placed in text files or libraries.

MVS: Compilation and Storage

On MVS, compile and link-edit the subroutine and store the module in a load library. If your subroutine calls other subroutines, compile and link-edit all the subroutines together in a single module.

If the subroutine is written in PL/I, include this link-editor control statement when link-editing the subroutine

```
ENTRY subroutine
```

where:

subroutine

Is the name of the subroutine.

Do not store the subroutine in the FUSELIB load library (FUSELIB.LOAD), as it may be overwritten when your site installs the next release of FOCUS.

Testing the Subroutine

Once you have successfully compiled your subroutine, access it and test it. In order to access the subroutine, you need to issue the GLOBAL command for CMS or the ALLOCATE command for MVS.

If an error occurs during your testing, check to see if the error is in the FOCUS request or in the subroutine. If you are uncertain about its source, apply this test:

1. Write a dummy subroutine that has the same arguments but only returns a constant.
2. Execute the request with the dummy subroutine.

   If the request executes the dummy subroutine normally, the error is in your subroutine. If the request still generates an error, the error is in the request.

If you intend to make your subroutine available to other users, be sure to document what your subroutine does, what the arguments are, what formats they have, and in what order they must appear in the FOCUS subroutine call.

Example of a Custom Subroutine: The MTHNAM Subroutine

This section illustrates how a subroutine can be written in FORTRAN, COBOL, PL/I, BAL Assembler, and C, and then executed in a FOCUS request. The subroutine, called MTHNAM, converts a number from 1 to 12 to the full name of the corresponding month (from January to December).

The subroutine performs the following:

1. The subroutine receives the input argument from the FOCUS request as a double-precision number.
2. It adds .000001 to the number. This compensates for rounding errors. (Rounding errors can occur since floating-point numbers are approximations and may be inaccurate in the last significant digit.)
3. It moves the number into an integer field.
4. If the number is less than 1 or greater than 12, it changes the number to 13.
5. It defines a 13-element array containing the names of the months. The last element is an error message.
6. It sets the index of the array equal to the number in the integer field. It then places the corresponding array element into the output argument. If the number is 13, the argument contains the error message.
7. It passes the output argument back to FOCUS.

The MTHNAM Subroutine Written in FORTRAN

This is a FORTRAN version of the MTHNAM subroutine. The fields are:

`MTH`

Is the double-precision number passed by FOCUS.

`MONTH`

Is the name of the month passed back to FOCUS. Since the character string 'September' contains nine letters, MONTH is a 3-element array. The subroutine passes the three elements back to FOCUS; FOCUS concatenates them into one field.

`A`

Is a 2-dimensional, 13 by 3 array containing the names of the months. The last three elements contain the error message.

`IMTH`

Is the integer representing the month.

The program is:

```
  SUBROUTINE MTHNAM (MTH,MONTH)
  REAL*8     MTH
  INTEGER*4  MONTH(3),A(13,3),IMTH
  DATA
 +     A( 1,1)/'JANU'/, A( 1,2)/'ARY '/, A( 1,3)/'    '/,
 +     A( 2,1)/'FEBR'/, A( 2,2)/'UARY'/, A( 2,3)/'    '/,
 +     A( 3,1)/'MARC'/, A( 3,2)/'H   '/, A( 3,3)/'    '/,
 +     A( 4,1)/'APRI'/, A( 4,2)/'L   '/, A( 4,3)/'    '/,
 +     A( 5,1)/'MAY '/, A( 5,2)/'    '/, A( 5,3)/'    '/,
 +     A( 6,1)/'JUNE'/, A( 6,2)/'    '/, A( 6,3)/'    '/,
 +     A( 7,1)/'JULY'/, A( 7,2)/'    '/, A( 7,3)/'    '/,
 +     A( 8,1)/'AUGU'/, A( 8,2)/'ST  '/, A( 8,3)/'    '/,
 +     A( 9,1)/'SEPT'/, A( 9,2)/'EMBE'/, A( 9,3)/'R   '/,
 +     A(10,1)/'OCTO'/, A(10,2)/'BER '/, A(10,3)/'    '/,
 +     A(11,1)/'NOVE'/, A(11,2)/'MBER'/, A(11,3)/'    '/,
 +     A(12,1)/'DECE'/, A(12,2)/'MBER'/, A(12,3)/'    '/,
 +     A(13,1)/'**ER'/, A(13,2)/'ROR*'/, A(13,3)/'*   '/
  IMTH=MTH+0.000001
  IF (IMTH .LT. 1 .OR. IMTH .GT. 12) IMTH=13
  DO 1 I=1,3
1 MONTH(I)=A(IMTH,I)
  RETURN
  END
```

The MTHNAM Subroutine Written in COBOL

This is a COBOL version of the MTHNAM subroutine. The fields are:

`MONTH-TABLE`

Is a field containing the names of the months and the error message.

`MLINE`

Is a 13-element array that redefines the MONTH-TABLE field. Each element (called A) contains the name of a month; the last element contains the error message.

`A`

Is one element in the MLINE array.

`IX`

Is an integer field that indexes MLINE.

`IMTH`

Is the integer representing the month.

`MTH`

Is the double-precision number passed by FOCUS.

`MONTH`

Is the name of the month passed back to FOCUS.

The program is:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MTHNAM.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
DATA DIVISION.
WORKING-STORAGE SECTION.
   01 MONTH-TABLE.
     05 FILLER PIC X(9) VALUE 'JANUARY  '.
     05 FILLER PIC X(9) VALUE 'FEBRUARY '.
     05 FILLER PIC X(9) VALUE 'MARCH    '.
     05 FILLER PIC X(9) VALUE 'APRIL    '.
     05 FILLER PIC X(9) VALUE 'MAY      '.
     05 FILLER PIC X(9) VALUE 'JUNE     '.
     05 FILLER PIC X(9) VALUE 'JULY     '.
     05 FILLER PIC X(9) VALUE 'AUGUST   '.
     05 FILLER PIC X(9) VALUE 'SEPTEMBER'.
     05 FILLER PIC X(9) VALUE 'OCTOBER  '.
     05 FILLER PIC X(9) VALUE 'NOVEMBER '.
     05 FILLER PIC X(9) VALUE 'DECEMBER '.
     05 FILLER PIC X(9) VALUE '**ERROR**'.
   01  MLIST REDEFINES MONTH-TABLE.
     05  MLINE OCCURS 13 TIMES INDEXED BY IX.
         10 A  PIC X(9).
   01  IMTH    PIC S9(5) COMP.
LINKAGE SECTION.
   01  MTH     COMP-2.
   01  MONTH   PIC X(9).
PROCEDURE DIVISION USING MTH, MONTH.
BEG-1.
      ADD 0.000001 TO MTH.
      MOVE MTH TO IMTH.
      IF IMTH < +1 OR > 12
        SET IX TO +13
      ELSE
        SET IX TO IMTH.
      MOVE A (IX) TO MONTH.
      GOBACK.
```

The MTHNAM Subroutine Written in PL/I

This is a PL/I version of the MTHNAM subroutine. The fields are:

`MTHNUM`

Is the double-precision number passed by FOCUS.

`FULLMTH`

Is the name of the month passed back to FOCUS.

`MONTHNUM`

Is the integer representing the month.

`MONTH_TABLE`

A 13-element array containing the names of the months. The last element contains the error message.

The program is:

```
MTHNAM:   PROC(MTHNUM,FULLMTH) OPTIONS(COBOL);
DECLARE  MTHNUM  DECIMAL FLOAT (16) ;
DECLARE  FULLMTH CHARACTER (9) ;
DECLARE  MONTHNUM FIXED BIN (15,0)  STATIC ;
DECLARE  MONTH_TABLE(13) CHARACTER (9)   STATIC
                        INIT  ('JANUARY',
                               'FEBRUARY',
                               'MARCH',
                               'APRIL',
                               'MAY',
                               'JUNE',
                               'JULY',
                               'AUGUST',
                               'SEPTEMBER',
                               'OCTOBER',
                               'NOVEMBER',
                               'DECEMBER',
                               '**ERROR**') ;
  MONTHNUM = MTHNUM + 0.00001 ;
  IF MONTHNUM < 1   MONTHNUM > 12 THEN
           MONTHNUM = 13 ;
  FULLMTH = MONTH_TABLE(MONTHNUM) ;
RETURN;
END MTHNAM;
```

The MTHNAM Subroutine Written in BAL Assembler

This is a BAL Assembler version of the MTHNAM subroutine.

```
         START 0
         STM   14,12,12(13)      save registers
         BALR  12,0              load base reg
         USING *,12
*
         L     3,0(0,1)          load addr of first arg into R3
         LD    4,=D'0.0'         clear out FPR4 and FPR5
         LE    6,0(0,3)          FP number in FPR6
         LPER  4,6               abs value in FPR4
         AW    4,=D'0.00001'     add rounding constant
         AW    4,DZERO           shift out fraction
         STD   4,FPNUM           move to memory
         L     2,FPNUM+4         integer part in R2
         TM    0(3),B'10000000'  check sign of original no
         BNO   POS               branch if positive
         LCR   2,2               complement if negative
*
POS      LR    3,2               copy month number into R3
         C     2,=F'0'           is it zero or less?
         BNP   INVALID           yes. so invalid
         C     2,=F'12'          is it greater than 12?
         BNP   VALID             no. so valid
INVALID  LA    3,13(0,0)         set R3 to point to item @13 (error)
*
VALID    SR    2,2               clear out R2
         M     2,=F'9'           multiply by shift in table
*
         LA    6,MTH(3)          get addr of item in R6
         L     4,4(0,1)          get addr of second arg in R4
         MVC   0(9,4),0(6)       move in text
*
         LM    14,12,12(13)      recover regs
         BR    14                return
*
```

```
          DS   0D                  alignment
 FPNUM    DS   D                   floating point number
 DZERO    DC   X'4E00000000000000' shift constant
 MTH      DC   CL9'dummyitem'      month table
          DC   CL9'JANUARY'
          DC   CL9'FEBRUARY'
          DC   CL9'MARCH'
          DC   CL9'APRIL'
          DC   CL9'MAY'
          DC   CL9'JUNE'
          DC   CL9'JULY'
          DC   CL9'AUGUST'
          DC   CL9'SEPTEMBER'
          DC   CL9'OCTOBER'
          DC   CL9'NOVEMBER'
          DC   CL9'DECEMBER'
          DC   CL9'**ERROR**'
          END  MTHNAM
```

The MTHNAM Subroutine Written in C

This is a C language version of the MTHNAM subroutine.

```
void mthnam(double *,char *);
void mthnam(mth,month)
double *mth;
char *month;
{
char *nmonth[13] = {"January  ",
                    "February ",
                    "March    ",
                    "April    ",
                    "May      ",
                    "June     ",
                    "July     ",
                    "August   ",
                    "September",
                    "October  ",
                    "November ",
                    "December ",
                    "**Error**"};
int imth, loop;
imth = *mth + .00001;
imth = (imth < 1 ¦¦ imth > 12 ? 13 : imth);
for (loop=0;loop < 9;loop++)
 month[loop] = nmonth[imth-1][loop];
}
```

The MTHNAM Subroutine Called by a FOCUS Request

The following example demonstrates how a FOCUS request uses the MTHNAM subroutine. The DEFINE command extracts the month portion of the pay date and executes the MTHNAM subroutine to convert it into the full name of the month. The name is stored in the PAY_MONTH field. The report request prints the monthly pay of Alfred Stevens.

The request is as follows:

```
DEFINE FILE EMPLOYEE
MONTH_NUM/M = PAY_DATE;
PAY_MONTH/A12 = MTHNAM (MONTH_NUM, PAY_MONTH);
END
TABLE FILE EMPLOYEE
PRINT PAY_MONTH GROSS
BY EMP_ID BY FIRST NAME BY LAST_NAME
BY PAY_DATE
IF LN IS STEVENS
END
```

This request produces the following report:

```
PAGE     1


EMP_ID     FIRST NAME    LAST_NAME     PAY_DATE      PAY_MONTH         GROSS
-------    ----------    ---------     --------      ---------         -----
071382660  ALFRED        STEVENS       81/11/30      NOVEMBER        $833.33
                                       81/12/31      DECEMBER        $833.33
                                       82/01/29      JANUARY         $916.67
                                       82/02/26      FEBRUARY        $916.67
                                       82/03/31      MARCH           $916.67
                                       82/04/30      APRIL           $916.67
                                       82/05/28      MAY             $916.67
                                       82/06/30      JUNE            $916.67
                                       82/07/30      JULY            $916.67
                                       82/08/31      AUGUST          $916.67
```

E Character Charts

Topics:

- Letters
- Numbers
- Punctuation
- Symbols
- Accent Marks and Accented Letters

This appendix contains charts listing the printable EBCDIC character set. EBCDIC is a code that enables IBM S/390 computers to store 256 characters as integers from 0 to 255. The character set in this appendix is based upon the display of an IBM 3270 terminal with the Hot Screen facility deactivated (SET SCREEN=OFF). Use of a 3270 emulator may give different results depending upon the code page used. The printable character sets are organized by topic:

- Letters: uppercase and lowercase.
- Numbers: integer.
- Punctuation: space/blank, common punctuation, brackets, and braces.
- Symbols: common, financial, mathematical, and rare.
- International: accent marks and accented letters.

In the charts, the printable characters appear on the left; their equivalent integers in decimal on the right.

Letters

Uppercase Letters	**Lowercase Letters**	**Uppercase Letters**	**Lowercase Letters**
A 193	a 129	N 213	n 149
B 194	b 130	O 214	o 150
C 195	c 131	P 215	p 151
D 196	d 132	Q 216	q 152
E 197	e 133	R 217	r 153

Uppercase Letters	Lowercase Letters	Uppercase Letters	Lowercase Letters
F 198	f 134	S 226	s 162
G 199	g 135	T 227	t 163
H 200	h 136	U 228	u 164
I 201	i 137	V 229	v 165
J 209	j 145	W 230	w 166
K 210	k 146	X 231	x 167
L 211	l 147	Y 232	y 168
M 212	m 148	Z 233	z 169

Numbers

Numbers	Numbers
0 240	5 245
1 241	6 246
2 242	7 247
3 243	8 248
4 244	9 249

Punctuation

Punctuation	Punctuation
space 64	' 125
. 75	“ 127
! 90	* 92
? 111	/ 97
, 107	(77
; 94	) 93
: 122	[65

Punctuation	Punctuation
- 96	] 66
_ 109	{ 192
	} 208

Symbols

Commonly-Used Symbols	Financial Symbols	Mathematical Symbols	Rarely-Used Symbols
@ 124	$ 91	+ 78	¤ 70
# 123	¢ 74	- 96	ß 71
% 108	£ 67	* 92	§ 72
& 80	¥ 68	/ 97	¦ 106
* 92		= 126	\ 224
° 81		@ 124	Æ 250
		¬ 95	æ 225
		| 79	Ø 251
		> 110	ø 234
		< 76	₧ 69

Accent Marks and Accented Letters

Accent Marks	Accent Marks
¯ 73	´ 85
ˇ 82	´ 86
^ 83	` 121
¨ 84	~ 161

Accented Letters	**Accented Letters**	**Accented Letters**	**Accented Letters**	**Accented Letters**
Á 203	À 158	Â 188	Ä 183	Ã 172
á 143	à 103	â 138	ä 117	ã 100
É 204	È 159	Ê 189	Ë 184	Ñ 220
é 144	è 104	ê 139	ë 118	ñ 157
Í 205	Ì 160	Î 190	Ï 185	Õ 173
í 154	ì 89	î 140	ï 119	õ 101
Ó 218	Ò 170	Ô 191	Ö 186	Å 252
ó 155	ò 98	ô 141	ö 120	å 235
Ú 219	Ù 171	Û 202	Ü 187	Ç 253
ú 156	ù 99	û 142	ü 128	ç 236
			ÿ 102	

Index

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. Send comments to:

Corporate Publications
Attn: Manager of Documentation Services
Information Builders
Two Penn Plaza
New York, NY 10121-2898

or FAX this page to (212) 967-0460, or call **Sara Elam** at (212) 736-4433, x**3207**.

Name: ____

Company: ____

Address: ____

Telephone: ______________________________ Date: ____________________

Comments:

Reader Comments