Information Builders.

# FOCUS® for S/390®

## Describing Data
Version 7.0

DN1001058.0599

# Contents

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 database administrators, application developers, or other information technology professionals who will create the metadata used by FOCUS to access corporate data.

# How This Manual Is Organized

This manual includes the following chapters:

| Chapter/Appendix | | Contents |
|---|---|---|
| **1** | *Describing Data With FOCUS* | Introduces Master Files and explains how to use them. |
| **2** | *Describing Data Sources* | Documents how to describe general aspects of your data source. |
| **3** | *Describing Groups of Fields* | Documents how to describe groups of related fields, or segments, of your data source. |
| **4** | *Describing Individual Fields* | Documents how to describe specific field-level information of your data source. |
| **5** | *Describing Sequential Data Files* | Provides supplementary information specific to sequential data files. |
| **6** | *Describing ISAM and VSAM Files* | Provides supplementary information specific to ISAM and VSAM files. |
| **7** | *Describing FOCUS Databases* | Provides supplementary information specific to FOCUS databases. |
| **8** | *Defining a Join in a Master File* | Describes how to create a relationship between two segments in separate Master Files that have at least one field in common. |
| **9** | *Checking and Changing Master Files: CHECK* | Describes how to use the CHECK command to validate your Master Files. |
| **10** | *Accessing FOCUS Databases: USE* | Describes how to assign a logical name to a FOCUS database. |
| **11** | *Providing File Security: DBA* | Describes how to control access to a data source by adding security attributes to the Master File. |
| **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** | *User Exits for Non-FOCUS Data Sources* | Describes how to read non-FOCUS data sources with user-written procedures. |
| **D** | *Rounding in FOCUS* | Describes how FOCUS rounds numbers for each numeric data type. |

# Summary of New Features

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

| New Feature | Version/ Release | Chapter |
|---|---|---|
| SET DATEDISPLAY = ON/OFF | 7.0.8R | Chapter 4, *Describing Individual Fields* |
| CHECK FILE HOLD ALL | 7.0.8R | Chapter 8, *Defining a Join in a Master File* |
| Displaying Invalid Smart Dates in Reports | 7.0.8R | Chapter 4, *Describing Individual Fields* |
| SET YRTHRESH = -n | 7.0.8R | Chapter 2, *Describing Data Sources*<br>Chapter 4, *Describing Individual Fields* |
| FORMAT = YYJUL | 7.0.8R | Chapter 4, *Describing Individual Fields* |
| Using Filters | 7.0.8 | Chapter 11, *Providing File Security: DBA* |
| Date Handling for the Year 2000 in FOCUS | 7.0.8 | Chapter 4, *Describing Individual Fields* |
| Cross-Century Dates in FOCUS Applications | 7.0.6 | Chapter 2, *Describing Data Sources* |
| Redefining Fields in Non-FOCUS Files | 7.0.5 | Chapter 5, *Describing Sequential Data Files* |

# 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. |
| `.`<br>`.`<br>`.` | 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 Online, 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 Describing Data With FOCUS

**Topics:**

- What Are FOCUS File Descriptions?
- How FOCUS Uses File Descriptions
- What Does a Master File Describe?
- Master File Syntax and Rules
- When to Create and Edit Master Files

FOCUS is a powerful application development language and decision support system that enables you to use a variety of data types in many operating environments. FOCUS makes this wide range of data available to you by means of its flexible data description language, which you can use with many different types of data, including:

- Relational data sources such as Oracle, DB2, SQL/DS, and Teradata.
- Sequential data files, both fixed-format and comma-delimited.
- The native FOCUS database.

You can also use the FOCUS data description language and related facilities to create data structures which are:

- **More complex.** For example, you can join different types of data sources together into a temporary structure from which you can report.
- **Simpler.** For example, you can define a logical view of your data in which only a subset of fields or columns is available to users.
- **Accessed differently.** If you are using a network database such as CA-IDMS, you can “rotate” the database to access the segments in a different order.

# What Are FOCUS File Descriptions?

When a FOCUS application accesses a data source, it needs to know how to interpret the data that it finds in the file. The application needs to know about:

- The overall structure of the data. For example, is the data source a relational database, a hierarchical database, or a simple sequential file? Depending upon the structure, how is it arranged or indexed?
- The specific data elements. For example, which fields are stored in the data source, and what is the data type of each field—alphanumeric, integer, date, or some other type?

An application can obtain the necessary information to interpret a data source by consulting a description of the data called a file description. The primary file description is called a Master File. For some types of data sources these Master Files are supplemented by related file descriptions called Access Files. You need to generate one Master File—and, for some types of data sources, one Access File—to describe each data source.

# How FOCUS Uses File Descriptions

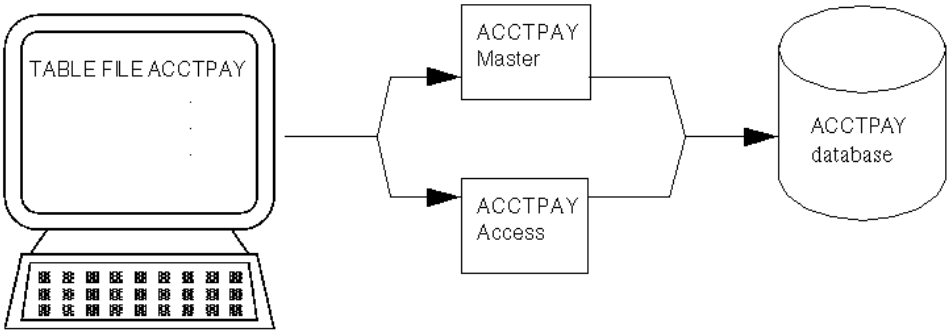
The Master and Access Files are stored as separate files apart from the data source. When FOCUS encounters a request—for example, a report request that begins with the command TABLE FILE ACCTPAY—it uses the Master and Access Files to interpret the data source in the following way:

**1.** Locates and reads the Master File. FOCUS identifies the Master File named in the request (ACCTPAY in this example). FOCUS then locates the Master File. If it is the one currently in memory (either in the Master File buffer or the LOAD command's buffer), FOCUS uses the memory image and then proceeds to Step 4. Otherwise, if it is not the FOCUS session's current Master File, FOCUS locates it on a storage device and loads it into memory.

**2.** Locates and reads the Access File. If this type of data source requires an Access File, FOCUS looks for one with the same name as the Master File. FOCUS then locates and reads the Access File as it located and read the Master File.

**3.** Reads the DBA security rules. If any DBA security has been specified for the data source, FOCUS reads the DBA information from the Master File. The DBA facility is described in Chapter 11, *Providing File Security: DBA*, or see the EDA documentation for your server.

4. Locates and reads the data source. FOCUS locates and reads the data source, interpreting its contents based on the description in the Master File and, for some types of data files, the Access File.




# What Does a Master File Describe?

The FOCUS Master File describes a data source at three levels:

- The type of data source.
- Groups of fields and how they relate to each other.
- Individual fields.

## The Data Source

The most important fact that FOCUS needs to know in order to interpret data, is what *kind* of data source it is. For example, is it a DB2 database, a FOCUS database, or a Teradata database?

### *Example* Describing a Data File

The following line is from a sample Master File—it identifies the name of the data source (EMPLOYEE) and its type (a FOCUS database):

```
FILENAME = EMPLOYEE, SUFFIX = FOC, $
```

# Groups of Fields

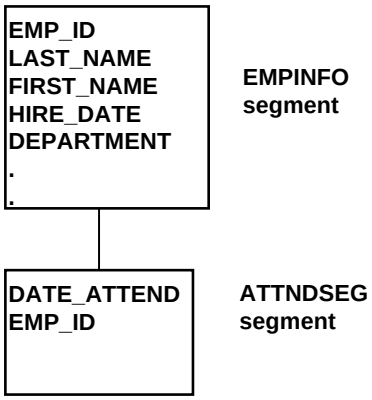
The Master File also identifies groups of fields and how they relate to each other. In most data sources, certain fields have a one-to-one correspondence: for each value of a field, the other fields will have exactly one corresponding value. These fields will form a group—in Master File terms, a segment; in relational terms, a table.

Related segments can be linked or joined together. You can even join segments from different *types* of data sources by issuing the JOIN command. For example, you could join two DB2 tables to a FOCUS database to a VSAM data file. The JOIN command is discussed in more detail in Chapter 8, *Defining a Join in a Master File*, and is fully documented in the *Creating Reports* manual.

***Example*** **Describing Groups of Fields**

For example, each employee has an identification number, a last name, a first name, and a department. In the EMPLOYEE database these fields form one segment.

You can join the EMPINFO segment to the ATTNDSEG segment, as shown in the following diagram, to track employee attendance at in-house training courses. Each employee can attend many courses, so there is a one-to-many relationship between an EMPINFO segment (representing one employee) and an ATTNDSEG segment (representing attendance in a course).



## Fields

To an application, the most important aspect of the data source is probably the data fields themselves. Every field has several characteristics, such as type of data and length or scale; some fields may be assigned additional characteristics, such as null data or online help information.

A Master File usually describes all of the fields in the file. In some cases, however, you can choose to create a logical view of the data in which only a subset of the fields are available, and then you would need to describe only those fields in your Master File.

### *Example* Describing Fields

For example, the following line from a sample Master File identifies a field's name (LAST_NAME), format (alphanumeric), length (15 characters), and the online help information available in a FOCUS application.

```
FIELDNAME = LAST_NAME,, FORMAT = A15,
    HELPMESSAGE = 'The last name can be up to 15 letters.', $
```

# Master File Syntax and Rules

This section describes the syntax and rules for a Master File and includes a sample Master File that illustrates them. The specifications for an Access File are similar, although the details may vary by type of data source; if your type of data source requires an Access File, see the appropriate Interface manual or, in a client/server environment, see the EDA documentation for your server for details.

### *Syntax* How to Create Master Files

A Master File describes a data file using a series of declarations:

1. A file declaration. In the sample EMPLOYEE Master File shown in the example that follows, note the FILENAME and SUFFIX file attributes.
2. A segment declaration for each segment. In the sample Master File, note the SEGNAME and SEGTYPE segment attributes.
3. A field declaration for each field. In the sample Master File, note the FIELDNAME, ALIAS, and FORMAT attributes. There are additional attributes that you can use to define a variety of field characteristics.

   Notice that the field declarations for all of a segment's fields follow the segment declaration.

Each declaration specifies a series of attributes in the form

*attribute = value*

where:

*attribute*
Is a Master File keyword that defines a file, segment, or field characteristic.

*value*
Is the value of the attribute.

**Note:** The attribute name must be in English; the attribute value can be in any supported national language.

Each attribute assignment is followed by a comma, and each field declaration is terminated with a dollar sign ($). Dollar signs are optional at the end of file and segment declarations.

For example,

```
FIELDNAME = LAST_NAME, ALIAS = LN, FORMAT = A15, $
```

is a field declaration; FIELDNAME, ALIAS, and FORMAT are field attributes and LAST_NAME, LN, and A15 are values. Notice the placement of commas and the final dollar sign.

## *Example* Sample Master File

The following sample Master File for the FOCUS database EMPLOYEE illustrates the syntax and rules for creating Master Files. For brevity, only the beginning of the Master File, describing the first two segments, is shown below. The numbers to the left of the Master File are shown for reference in *How to Create Master Files* on page 1-5 and do not appear in the actual Master File.

```
1.  FILENAME = EMPLOYEE, SUFFIX = FOC, $

2.  SEGNAME = EMPINFO, SEGTYPE = S1, $
3.      FIELDNAME = EMP_ID       ,ALIAS = EID    ,FORMAT = A9      ,$
3.      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    ,$

2.  SEGNAME = FUNDTRAN, SEGTYPE = U, PARENT = EMPINFO, $
3.      FIELDNAME = BANK_NAME    ,ALIAS = BN     ,FORMAT = A20     ,$
3.      FIELDNAME = BANK_CODE    ,ALIAS = BC     ,FORMAT = I6S     ,$
        FIELDNAME = BANK_ACCT    ,ALIAS = BA     ,FORMAT = I9S     ,$
        FIELDNAME = EFFECT_DATE  ,ALIAS = EDATE  ,FORMAT = I6YMD   ,$
```

## Improving Readability

You can begin each attribute assignment in any column that you wish; this makes it easy for you to indent segment or field declarations. To position text, use blank spaces, not the Tab key.

You can include blank spaces between the attribute, value, equal sign (=), comma, and dollar sign ($) to make the Master File easier for you to read. You can also include blank lines to separate segment or field declarations. FOCUS does not require any blank spaces or lines and ignores them if they are there. For example, the following lines from a sample Master File are valid:

```
SEGNAME = EMPINFO, SEGTYPE=S1, $
    FIELDNAME = EMP_ID, ALIAS= EID, FORMAT = A9, $
```

You can spread a declaration across as many lines as you wish. For example, for a given declaration you can put each attribute assignment on a separate line, combine several on each line, or include the entire declaration on a single line (if it will fit). Each line can be up to 80 columns long. For example:

```
FIELDNAME = MEMBERSHIP, ALIAS = BELONGS, FORMAT = A1, MISSING = ON,
    DESCRIPTION = This field indicates the applicant's membership status,
    ACCEPT = Y OR N, FIELDTYPE = I,
    HELPMESSAGE = 'Please enter Y for Yes or N for No', $
```

## Using Comments

You can add a comment to any declaration, typing the comment in the columns following the terminating dollar sign. You can even create an entire comment line by inserting a new line following a declaration and placing a dollar sign at the beginning of the line. (Note that adding a comment line will terminate the previous declaration if it has not already been terminated.) FOCUS ignores everything on a line following the dollar sign. For example:

```
FILENAME = EMPLOYEE, SUFFIX = FOC, $ This is a comment.
$ This entire line is a comment!
SEGNAME = EMPINFO, SEGTYPE = S1, $
```

**Note:** To comment a line in the DBA section of the Master File, follow the above rules; do not include a comma dollar sign (,$) on a line by itself in the Master File.

## Using Attribute Names

You can refer to any attribute by its standard name, its alias, or its shortest unique truncation. For example, the following two attribute assignments are identical:

```
FILENAME = EMPLOYEE
FILE = EMPLOYEE
```

# When to Create and Edit Master Files

Before you can use FOCUS with a data source—for example, to report from the data source, or to update it—a Master File needs to exist for the file. If there is no Master File (and for some types of data file, no Access File), you need to generate it, using either one of FOCUS' automatic tools or a text editor. Once the Master File exists, it never needs to be altered unless someone wishes to change the structure of the data source (for example, by adding a new field, or lengthening an existing field). If you edit the Master File, you can then change the data source accordingly.

## Creating Master Files

You can generate FOCUS Master Files in several ways:

- Manually, using a standard text editor, such as XEDIT, SPF, or the FOCUS text editor, TED. You can code the Master File yourself based on the information in this manual.

  For information about using the FOCUS text editor, TED, see the *Overview and Operating Environments* manual.

  After editing a Master File, you should issue the CHECK FILE command to refresh your session's image of the Master File.

- Automatically, using the FOCUS AUTO facilities or the COBOL to FOCUS Translator. If you are using a data source that already has its own native file description—for example, a DB2 database with its own relational schema—then you can run the AUTODB2 facility to automatically generate a Master and Access File based on the original schema.

  To find out if an AUTO facility exists for the type of data source you are using, see the appropriate Interface manual or, in a client/server environment see the EDA documentation for your server.

## Checking Master Files

You can use the FOCUS CHECK command when you design a data file and develop a Master File. CHECK displays a diagram illustrating the structure of the data view that corresponds to the Master File. You can also use CHECK to generate reports about the information in the Master File and to indicate in what order information will be retrieved from the data source when you run a FOCUS report against it.

# 2 Describing Data Sources

**Topics:**

- Specifying the Data Source Name: FILENAME
- Specifying the Data Source Type: SUFFIX
- Sliding Window File Attributes for Handling Two-Digit Years

The most important fact that FOCUS needs to know in order to interpret the data in the data source is what *kind* of data source it is. For example, is it a DB2 database, a FOCUS database, or a Teradata database?

Your Master File needs to identify the name and type of your data source. There are two attributes used to do this:

- FILENAME, which identifies the name of the data source.
- SUFFIX, which identifies the type of data source.

You can optionally specify a hundred-year window that assigns a century value to dates stored with two-digit years. There are two attributes that do this:

- FDEFCENT, which identifies a century.
- FYRTHRESH, which identifies the year value that is the lower limit for FDEFCENT.

The following line from a sample Master File identifies the name of the data source (EMPLOYEE) and its type (a FOCUS database):

```
FILENAME = EMPLOYEE, SUFFIX = FOC, $
```

# Specifying the Data Source Name: FILENAME

FILENAME is the first attribute specified in a Master File. It is recommended that you use it to document the name of the data source that the Master File is describing.

*Syntax*

## How to Specify the Data Source Name

The syntax of the FILENAME attribute is

```
FILE[NAME] = data_file_name
```

where:

*data_file_name*

Is the name of the data source that this Master File describes. It can be up to eight characters long.

The first character should be a letter, and the remaining characters can be any combination of letters, numbers, and underscores (_). Other characters are not recommended and may cause problems in some operating environments or when resolving expressions.

*Example*

## Specifying the Data Source Name

For example, if the data source described by the Master File is the EMPLOYEE database, you could use the FILENAME attribute in the following way:

```
FILENAME = EMPLOYEE
```

*Reference*

## Usage Notes for FILENAME

Note the following rules when using FILENAME:

- **Alias.** FILENAME has an alias of FILE.
- **Changes.** We recommend that FILENAME always be set to the file name of the data source. For a FOCUS database with no cross-referenced files you can change the file name and rename the Master File and database without rebuilding the database. If any segment is cross-referenced, all other Master Files that reference this file must be changed to reflect the change in the file name.

# Specifying the Data Source Type: SUFFIX

The SUFFIX attribute identifies the type of data source you are using—for example, a DB2 database or a FOCUS database. FOCUS then knows which Interface to apply when it accesses the data source.

## *Syntax* How to Specify the Data Source Type

The syntax of the SUFFIX attribute is

`SUFFIX = filetype`

Valid values are:

`FIX`
: Indicates that the data source is a fixed-format sequential data file. This is the default value.

`COM`
: Indicates that the data source is a free-format—that is, comma-delimited—sequential data file.

`FOC`
: Indicates that the data source is a FOCUS database.

`PRIVATE`
: Indicates that a customer-coded interface is being used to read the data source.

You can also use the SUFFIX attribute to identify many relational and non-relational data sources:

| Types of Files | Suffixes |
|---|---|
| ADABAS | `ADBSIN` |
| CA-DATACOM/DB | `DATACOM` |
| CA-IDMS/DB | `IDMSR` |
| DB2 | `DB2 or SQLDS` |
| IMS | `IMS` |
| ISAM | `ISAM` |
| Millennium | `CPMILL2 or CPMILL3` |
| MODEL 204 | `M204IN` |
| SQL/DS | `SQLDS` |
| Oracle | `SQLORA` |
| SUPRA | `SUPRA` |
| SYSTEM 2000 | `S2K` |
| Teradata | `SQLDBC` |
| TOTAL | `TOTIN` |
| VSAM (entry or key sequenced) | `VSAM` |

For more information, see the appropriate Interface manual.

***Example*** **Specifying the Data Source Type**

For example, if the data source described by the Master File is a fixed-format data file, you would use the SUFFIX attribute in the following way:

```
SUFFIX = FIX
```

You can also create your own data access module. Techniques for identifying and using your own data access routines are described in Appendix C, *User Exits for Non-FOCUS Data Sources*.

*Reference*

### Usage Notes for SUFFIX

Note the following rules when using SUFFIX:

- **Alias.** SUFFIX has an alias of FILESUFFIX.
- **Changes.** Because each type of data source has only one correct SUFFIX value, you cannot change the SUFFIX value.

## Sliding Window File Attributes for Handling Two-Digit Years

If your data source contains date values stored with two-digit years, you can use the FDEFCENT and FYRTHRESH attributes to assign century values to these fields. See the *Developing Applications* manual for a description of the sliding window approach and for related SET command parameters. You can also set sliding window attributes at the field level as described in *Sliding Window Field Attributes for Handling Two-Digit Years* in Chapter 4, *Describing Individual Fields*.

*Syntax*

### How to Establish Cross-Century Dates in a Master File

At the file level, the settings FDEFCENT and FYRTHRESH describe a 100-year window. The FDEFCENT syntax is

```
{FDEFCENT|FDFC} = cc
```

where:

*cc*
: Is a two-digit century value. The default value is 19.

The FYRTHRESH syntax is

```
{FYRTHRESH|FYRT} = {yy|-nn}
```

where:

*yy*
: Is the year for which the century value changes to FDEFCENT. Below this value, the century is FDEFCENT+1. The default value is zero.

*-nn*

Sets FYRTHRESH to an offset from the current year. The default value is zero.

You decide the number of years to offset in FYRTHRESH. For example, if the current year is 1998 and you wish to set FYRTHRESH to 60, you can set FYRTHRESH to -38 (1998 - 38 = 60). By setting FYRTHRESH to a negative number FOCUS subtracts that number from the current year. In the year 1999, FYRTHRESH is 61 instead of 60 (1999 - 38 = 61) illustrating how the moving window functions without outside intervention.

If you set FYRTHRESH to a value that crosses a century boundary, the value of FDEFCENT is recalculated. For example, if you set FYRTHRESH to minus 99 (1998-99 = -1), FDEFCENT is calculated to 18 and FYRTHRESH becomes 99. The 100-year span begins with a pivot year of 1899 and ends with year 1998. ? SET and ? SET ALL reflect the FDEFCENT setting. **Note:** When you use this option, do not specify FDEFCENT. It is calculated automatically.

## *Example* Establishing Cross-Century Dates in a Master File

```
FILENAME = EMPLOYEE, SUFFIX = FOC, FDEFCENT = 20, FYRTHRESH = 66,$
 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,$
```

To see the application of FDEFCENT and FYRTHRESH to interpret two-digit years, consider the following:

```
FDEFCENT = 19, FYRTHRESH = 80
```

describes a range from 1980 to 2079. If a two-digit year field contains 99, the year is 1999. If it is 79, the year is 2079. If it is 00, the year is 2000.

## *Reference* Special Considerations

- Field level settings take precedence over file level settings.
- Compiled MODIFYs from releases earlier than Version 7.0 Release 6 must be recompiled to use this feature.
- ON TABLE SET is not supported.
- In HOLD Master Files, the concatenation of FILE and FIELD level information is propagated as FIELD level.
- With DEFINE and COMPUTE, both DFC and YRT must be specified. Each can be specified separately in the Master File.

# 3 Describing Groups of Fields

**Topics:**

- Defining a Single Group of Fields
- Identifying a Logical View: Redefining a Segment
- Relating Multiple Groups of Fields
- Logical Dependence: The Parent-Child Relationship
- Logical Independence: Multiple Paths
- Cardinal Relationships Between Segments
- One-to-One Relationships
- One-to-Many Relationships
- Many-to-Many Relationships
- Recursive Relationships
- Relating Segments From Different Types of Data Sources
- Rotating a Database: Alternate Views

In virtually every data file, certain fields have a one-to-one correspondence and form a group. You can relate different groups to each other. For some types of data files you can even define a logical view of a group—that is, a subset. You identify these groups to FOCUS using attributes in the Master and Access Files. You identify the relationships between groups using attributes in the Master and Access Files, as well as related facilities such as the JOIN command.

Defining a group of fields is described in *Defining a Single Group of Fields* on page 3-2. Relating these groups to each other is described in *Relating Multiple Groups of Fields* on page 3-7.

These sections describe these general concepts and explain how to implement them using Master File attributes. If your type of data source also requires an Access File, see the appropriate Interface manual for supplementary information about defining groups and group relations within the Access File.

# Defining a Single Group of Fields

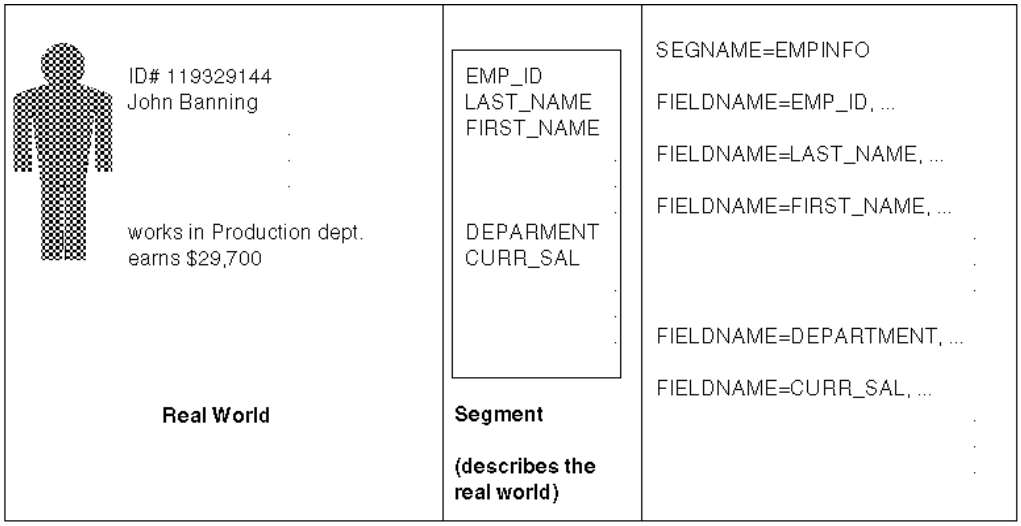
In most data sources, certain fields have a one-to-one correspondence: for each value of a field, the other fields will have exactly one corresponding value. For example, consider the standard EMPLOYEE database:

- Each employee has one ID number and the number is unique to that employee.
- For each ID number—that is, for each employee—there is one first and last name, one date hired, one department, and one current salary.

In the database, each of these employee characteristics is represented by a field. The employee is represented by the group of fields. In FOCUS terms, this group is called a segment.

## Understanding Segments

While the term *segment* may not be familiar to you, the concept behind it is universal: it is a group of fields that have a one-to-one correspondence with each other and usually describe a group of related characteristics. For example, in a relational database, a segment is equivalent to a table.

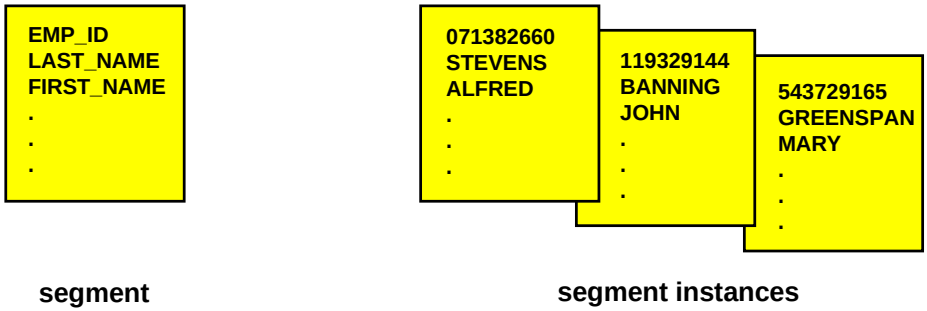



Segments are the building blocks of larger data structures. You can relate different segments to each other, and describe the new structures to FOCUS, as described in *Relating Multiple Groups of Fields* on page 3-7.

## Understanding Segment Instances

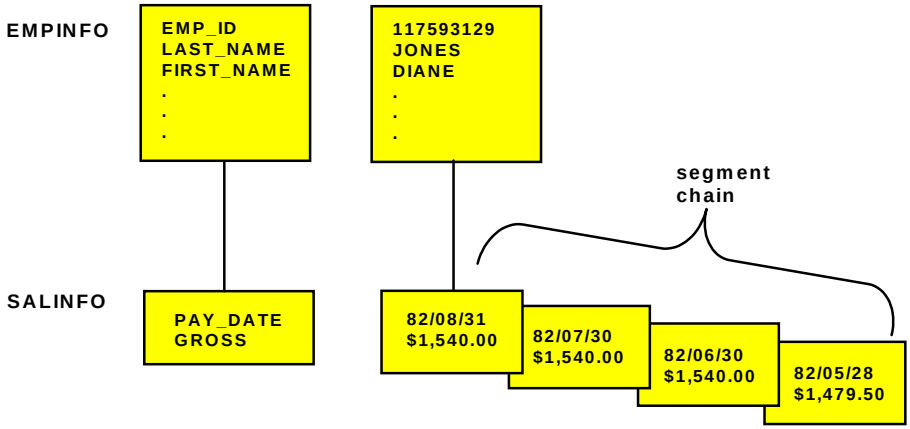
While a segment is abstract—a description of data—the segment instances that correspond to it are the actual data: each instance is an occurrence of segment values found in the data file. For users of a relational database, an instance is equivalent to a row in a table. In a single segment file, a segment instance is the same as a record.

The relationship of a segment to its segment instances is illustrated in the following diagram:




## Understanding Segment Chains

All of a segment's instances that are descended from a single parent instance are collectively known as a segment chain. In the special case of a root segment, which of course has no parent, all of the root instances form a chain. (The parent-child relationship is discussed in *Logical Dependence: The Parent-Child Relationship* on page 3-9.)




You describe a segment to FOCUS using the SEGNAME and SEGTYPE attributes in the Master File. The SEGNAME attribute is described in *Identifying the Segment: SEGNAME* on page 3-4.

# Identifying Key Fields

Most segments also have keys fields—that is, one or more fields that uniquely identify each segment instance. In the EMPLOYEE example, the ID number is the key because each employee has one ID number, and no other employee has the same number. The ID number is represented in the database by the EMP_ID field.

If your data source uses an Access File, you may need to specify which fields serve as keys by identifying them in the Access File. If the data source also happens to be a relational database, then within the Master File, the fields constituting the primary key should be the first fields described for that segment—that is, their field declarations should come before any others in that segment.

For FOCUS databases, you identify key fields and their sorting order using the SEGTYPE attribute in the Master File, as shown in Chapter 7, *Describing FOCUS Databases.* You position the key fields as the first fields within their segment.

# Identifying the Segment: SEGNAME

SEGNAME is the first attribute specified in a segment declaration and is used to identify the segment.

You can give the segment any name consisting of up to eight characters. You will probably want to make the Master File self-documenting by setting SEGNAME to something meaningful to the user or the native file manager. For example, if you are describing a DB2 table, you might want to assign the table name (or an abbreviation) to SEGNAME.

If your data source uses an Access File as well as a Master File, you must specify the same segment name in both.

## *Syntax* How to Identify a Segment

The syntax of the SEGNAME attribute is

```
SEGNAME = segment_name
```

where:

*segment_name*

Is the name you want to use to identify this segment. It can be up to eight characters long.

The first character must be a letter, and the remaining characters can be any combination of letters, numbers, and underscores (_). Other characters are not recommended and may cause problems in some operating environments or when resolving expressions.

### *Example* Identifying a Segment

For example, if the segment corresponded to a relational table named TICKETS, and you wanted to give the segment the same name, you could use the SEGNAME attribute in the following way:

```
SEGNAME = TICKETS
```

### *Reference* Usage Notes for SEGNAME

Note the following rules when using SEGNAME:

- **Uniqueness.** Within a Master File, each segment name must be unique.

  The only exception to this rule is within a FOCUS database, where cross-referenced segments in Master File defined joins can have the same name as other cross-referenced segments in Master File defined joins. If cross-referenced segments do have identical names, you can still refer to them uniquely by using the CRSEGNAME attribute. See Chapter 8, *Defining a Join in a Master File*, for more information.

- **Alias.** SEGNAME has an alias of SEGMENT.

- **Changes.** In a FOCUS database, you cannot change the value of SEGNAME once data has been entered into the database. For all other types of data files, you can change SEGNAME as long as you also change all references to it—for example, any references in the Master and Access File.

## Identifying a Logical View: Redefining a Segment

The segments that you define to FOCUS usually correspond to underlying groups in your data file. For example, a segment could be a table in a relational database.

However, you are not limited to using the segment as it was originally defined in the native data source: you can define a logical view to FOCUS in which you include only a subset of the segment's fields (similar to a relational view), or else define the unwanted fields as one or more "filler" fields. This technique can be helpful if, for example, you only want to make some of the segment's fields available to an application or its users. Of course, if you want to explicitly restrict access at the file, segment, or field level based on userid, field values, and other characteristics, you can use the DBA facility, as described in Chapter 11, *Providing File Security: DBA*.

You can use the following methods with the following types of data files:

- **Relational databases.** You can omit unwanted fields from the segment description.

- **Sequential data files.** You can define unwanted fields as one or more filler fields.

- **FOCUS databases.** You can define unwanted fields as one or more filler fields.

## Omitting Fields: Creating a Segment Subset

You can define a logical view for a relational database by omitting the unwanted fields from the segment's description in the Master File. For example, consider the following Master File for an Oracle table named EMPFACTS:

```
FILENAME = EMPFACTS, SUFFIX = SQLORA

SEGNAME = EMPFACTS, SEGTYPE = S0, $
    FIELDNAME = EMP_NUMBER,   ALIAS = ENUM,   FORMAT = A9,     ACTUAL = A9,  $
    FIELDNAME = LAST_NAME,    ALIAS = LNAME,  FORMAT = A15,    ACTUAL = A15, $
    FIELDNAME = FIRST_NAME,   ALIAS = FNAME,  FORMAT = A10,    ACTUAL = A10, $
    FIELDNAME = HIRE_DATE,    ALIAS = HDT,    FORMAT = I6YMD,  ACTUAL = DATE,$
    FIELDNAME = DEPARTMENT,   ALIAS = DPT,    FORMAT = A10,    ACTUAL = A10, $
    FIELDNAME = SALARY,       ALIAS = SAL,    FORMAT = D12.2M,ACTUAL = D8,  $
    FIELDNAME = JOBCODE,      ALIAS = JCD,    FORMAT = A3,     ACTUAL = A3,  $
    FIELDNAME = OFFICE_NUM,   ALIAS = OFN,    FORMAT = I8,     ACTUAL = I4,  $
```

If you develop an application that refers to only an employee's ID and name, and you want this to be reflected in the application's view of the segment, you could code an alternative Master File that names only the desired fields:

```
FILENAME = EMPFACTS, SUFFIX = SQLORA

SEGNAME = EMPFACTS, SEGTYPE = S0, $
    FIELDNAME = EMP_NUMBER,   ALIAS = ENUM,   FORMAT = A9,     ACTUAL = A9 ,$
    FIELDNAME = LAST_NAME,    ALIAS = LNAME,  FORMAT = A15,    ACTUAL = A15,$
    FIELDNAME = FIRST_NAME,   ALIAS = FNAME,  FORMAT = A10,    ACTUAL = A10,$
```

## Redefining Fields: Creating a Filler Field

You can define a logical view for certain data sources such as a sequential or FOCUS data source simply by defining the fields excluded from the view as one or more "filler" fields. You would define the field's format to be alphanumeric, its length to be the number of bytes making up the underlying fields, and its name and alias to be blank. (Field declarations and length are discussed in detail in Chapter 4, *Describing Individual Fields*.)

For example, consider the EMPINFO segment of the EMPLOYEE database:

```
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,        $
```

If you develop an application that refers to only an employee's ID and name, and you want this to be reflected in the application's view of the segment, you could code an alternative Master File that explicitly names only the desired fields:

```
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 = ,             ALIAS = ,          FORMAT = A29,         $
```

Note that the filler field is defined as an alphanumeric field of 29 bytes, which is the combined internal length of the fields it replaces: HIRE_DATE (4 bytes), DEPARTMENT (10 bytes), CURR_SAL (8 bytes), CURR_JOBCODE (3 bytes), and ED_HRS (4 bytes).

# Relating Multiple Groups of Fields

Once you have described groups of fields—that is, segments—to FOCUS, you can relate them to each other to build more sophisticated data structures. You can:

- **Describe physical relationships.** If groups of fields are already physically related in your data file, you can describe the relationship to FOCUS.
- **Describe logical relationships.** You can describe a logical relationship between any two segments that have at least one field in common by joining them. The underlying data structures remain physically separate, but FOCUS treats them as if they were part of a single structure. The new structure can include segments from the same or different types of data sources.

  Note that if you are creating a new FOCUS database, you can implement segment relationships in several ways, depending upon your design goals, as described in Chapter 7, *Describing FOCUS Databases.*

To describe a data structure containing several segments—whether it is a multi-segment data source or several data sources that have been joined together—you need to be aware of two simple concepts:

- Logical dependence between related segments.
- Logical independence between unrelated segments.

# Facilities for Specifying Relationships

FOCUS provides several facilities for specifying relationships between segments. The use of a Master and Access File to specify a relationship is fully documented in this chapter. The JOIN command, which joins segments into a structure from which you can report, is fully described in the *Creating Reports* manual.

Note that a related facility, the MATCH FILE command, enables you to implement many different types of sophisticated relationships by first describing the relationship as a series of extraction and merging conditions and then merging the related data into a new single-segment data file. The result is not a joined structure but an entirely new file that you can process further. The original data sources themselves remain unchanged. The MATCH FILE command is documented in the *Creating Reports* manual.

# Identifying the Parent Segment: PARENT

You identify a segment's parent by using the PARENT attribute in the segment declaration of the Master File. Because a root segment has no parent, you do not specify the PARENT segment when declaring a root.

Note that a parent segment must be declared in the Master File before any of its child segments.

## *Syntax* How to Identify the Parent Segment

The syntax of the PARENT attribute is

```
PARENT = segment_name
```

where:

*segment_name*
Is the name of the segment's parent as previously declared in the Master File.

## *Example* Identifying a Parent Segment

For example, in the EMPLOYEE database, DEDUCT's parent is SALINFO, and so the segment declaration for DEDUCT includes the following attribute:

```
PARENT = SALINFO
```

### *Reference* Usage Notes for PARENT

Note the following rules when using PARENT:

- **Default.** If you do not specify the PARENT attribute, it defaults to the value of the most recently specified segment. If the PARENT attribute has not been specified in any prior segment declarations in this Master File, the previous segment becomes the parent.

  The PARENT attribute is recommended for unique segments with a SEGTYPE of U.

- **Alias.** PARENT has no alias.

- **Changes.** If the parent-child relationship is permanently implemented within the structure of the data source, as, for example, within a FOCUS database, then you cannot change the parent attribute without changing the underlying structure of the data source. However, if the parent-child relationship is temporary, as, for example, when you join several relational tables in the Master File, then you can change the PARENT attribute.

## Identifying the Type of Relationship: SEGTYPE

The SEGTYPE attribute specifies the type of relationship that a segment has to its parent. SEGTYPE is part of the segment declaration and is used in different ways with different types of data sources. For FOCUS databases, see Chapter 7, *Describing FOCUS Databases.* For sequential data files see Chapter 5, *Describing Sequential Data Files*. For ISAM and VSAM files, see Chapter 6, *Describing ISAM and VSAM Files*. For other types of data sources, see the appropriate Interface manual for details.

# Logical Dependence: The Parent-Child Relationship

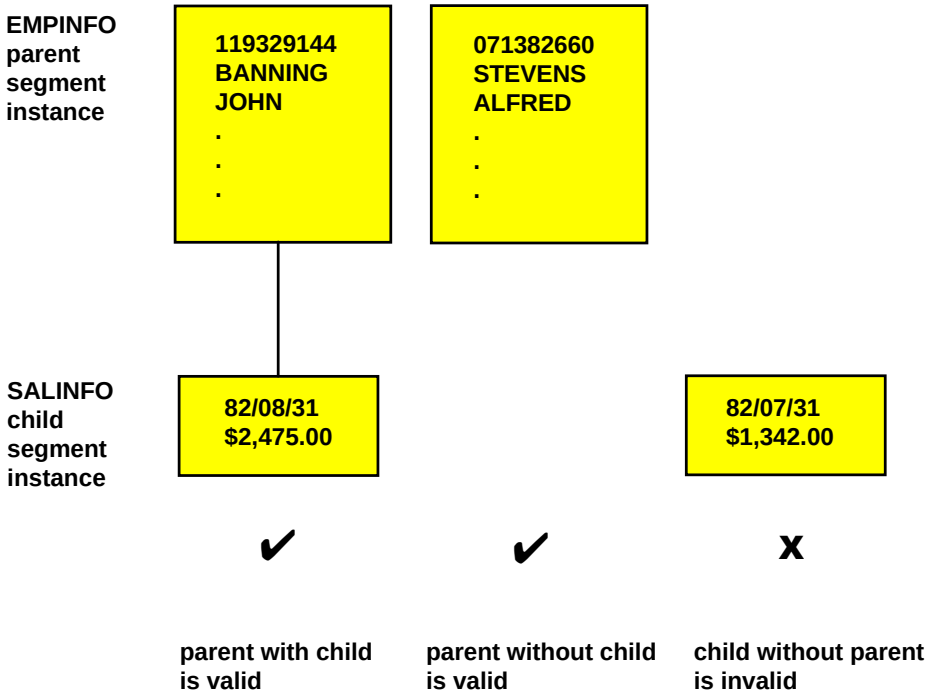
Logical dependence between segments is expressed in terms of the parent-child relationship: a child segment is dependent upon its parent segment. This means that an instance of the child segment can exist only if a related instance of the parent segment exists. The parent segment has logical precedence in the relationship, and is retrieved first when FOCUS accesses the data file.

Note that if the parent-child relationship is logical and not physical—that is, if it is implemented as a join—it is possible to have a child instance without a related parent instance. In this case, the child instance will not be accessible through the join, although, of course, it will still be accessible independently.

If the parent and child segments are related by a join, the parent is also known as the host segment, and the child is also known as the cross-referenced segment. The fields on which the join is based—that is, the matching fields in the host and cross-referenced segments—are known respectively as the host and cross-referenced fields.

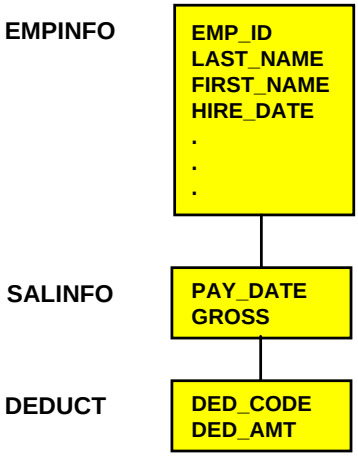
*Example* **A Simple Parent-Child Relationship**

For example, in the EMPLOYEE database, the EMPINFO and SALINFO segments are related: EMPINFO identifies an employee by ID number, while SALINFO contains the employee's pay history. EMPINFO is the parent segment, and SALINFO is a child segment dependent upon it. This relationship is illustrated by the fact that it is possible to have an employee identified by ID and name for whom no salary information has been entered—that is, the parent instance without the child instance; but it is meaningless to have salary information for an employee if we do not know who the employee is—that is, a child instance without the parent instance.



*Example* **Parent-Child Relationships With Multiple Segments**

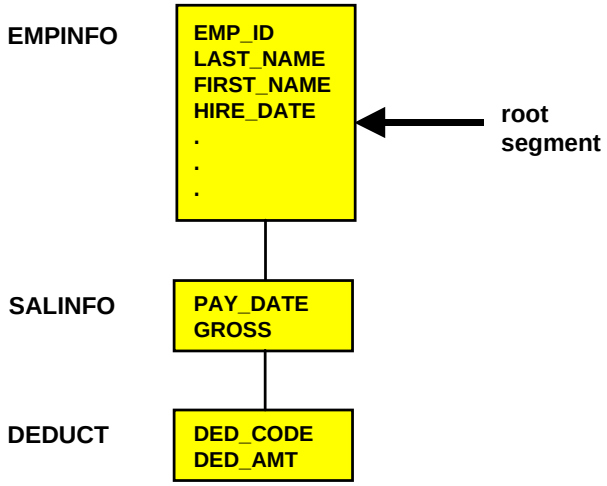
The same general parent-child relationships hold for data structures containing more than two segments. For example, consider the following diagram of a portion of the EMPLOYEE database, containing the EMPINFO, SALINFO, and DEDUCT segments. DEDUCT contains payroll deduction information for each paycheck.




EMPINFO is related to SALINFO, and in this relationship EMPINFO is the parent segment and SALINFO is the child segment. SALINFO is also related to DEDUCT; in this second relationship, SALINFO is the parent segment and DEDUCT is the child segment. Just as SALINFO is dependent upon EMPINFO, DEDUCT is dependent upon SALINFO.

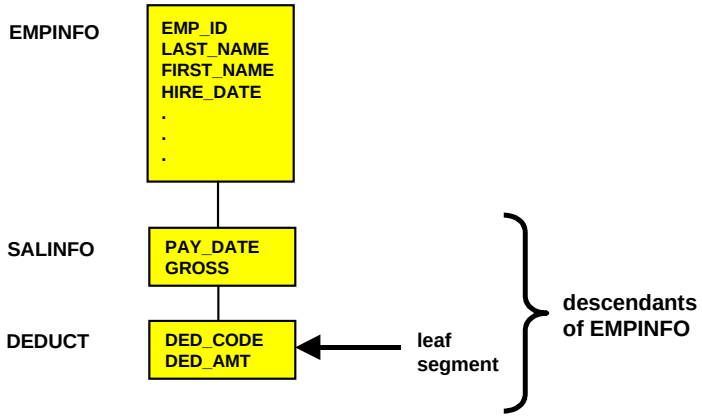
## Understanding Root Segments

The segment that has logical precedence over the entire data structure—in a sense, the "parent" of the entire structure—is called the root segment. (The term *root* is used because a data structure can branch like a tree, and the root segment, like the root of a tree, is the source of the structure.) In this example, EMPINFO is the root: it has no parent, and all other segments in the structure are its children directly (SALINFO) or indirectly (DEDUCT).



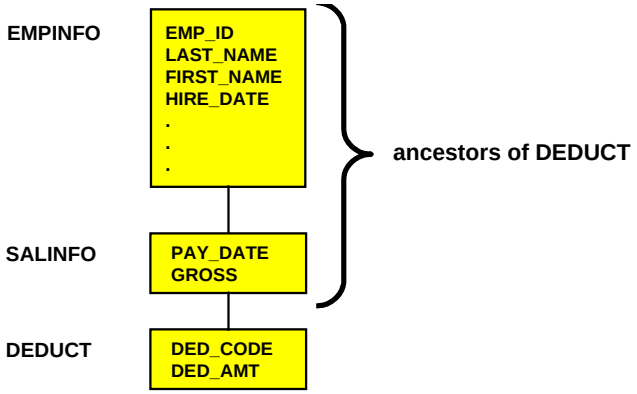

## Understanding Descendant Segments

We refer to a segment's direct and indirect children collectively as its descendant segments. SALINFO and DEDUCT are descendants of EMPINFO; DEDUCT is also a descendant of SALINFO. A descendant segment that has no children is called a leaf segment (because the branching of the data structure "tree" ends with the leaf). DEDUCT is a leaf.



# Understanding Ancestral Segments

We refer to a segment's direct and indirect parents as ancestral segments. In our example, SALINFO and EMPINFO are ancestors of DEDUCT.

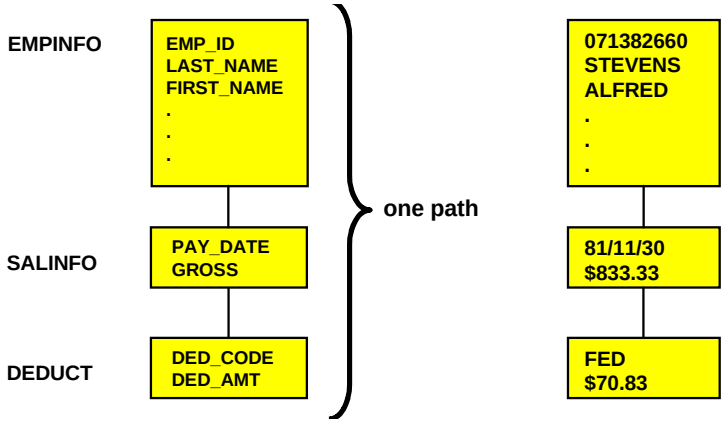


# Logical Independence: Multiple Paths

A group of segments that are related to each other as a sequence of parent-child relationships, beginning with the root segment and continuing down to a leaf, is called a path. Because the path is a sequence of parent-child relationships, each segment is logically dependent upon all of the segments higher in the path.

## Understanding a Single Path

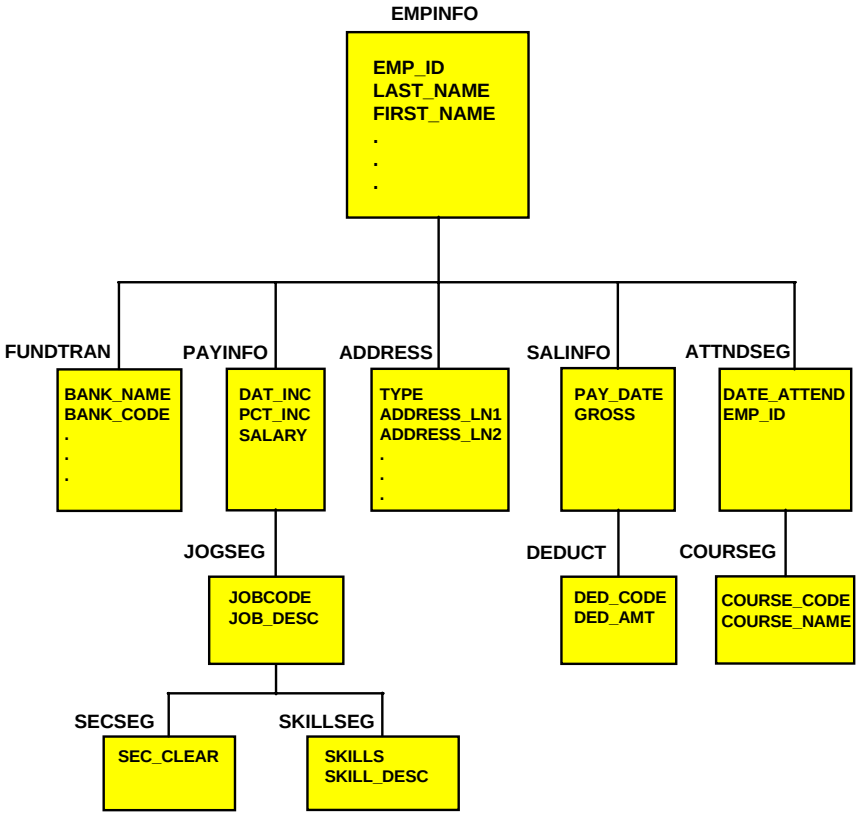
For example, in the following view of the EMPLOYEE database, EMPINFO, SALINFO, and DEDUCT form a path. An instance of DEDUCT (paycheck deductions) can exist only if a related instance of SALINFO (the paycheck) exists, and the instance of SALINFO (the employee's paycheck) can exist only if a related instance of EMPINFO (the employee) exists.



# Understanding Multiple Paths

Now consider the full EMPLOYEE structure, which includes the EMPLOYEE database and the JOBFILE and EDUCFILE databases that have been joined to it.

This is a multi-path data structure: there are several paths, each beginning with the root segment and ending with a leaf. Every leaf segment in a data structure is the end of a separate path.




EMPINFO | FUNTRAN

EMPINFO | PAYINFO | JOBSEG | SECSEG

EMPINFO | PAYINFO | JOBSEG | SKILLSEG

EMPINFO | ADDRESS

EMPINFO | SALINFO | DEDUCT

EMPINFO | ATTNDSEG | COURSEG

## Understanding Logical Independence

The EMPLOYEE data structure has six paths. The paths begin with the EMPINFO segment (the root), and end with:

- The FUNDTRAN segment
- The SECSEG segment
- The SKILLSEG segment
- The ADDRESS segment
- The DEDUCT segment
- The COURSEG segment

Each path is logically independent of the others. For example, an instance of DEDUCT is dependent upon its ancestor segment instances SALINFO and EMPINFO; but the ADDRESS segment lies in a different path, and so DEDUCT is independent of ADDRESS.

This is because an employee's deductions are identified by the paycheck from which they came, so deduction information can be entered into the database only if the paycheck from which the deduction was made was entered first. However, deductions are not identified by the employee's address; an employee's paycheck deduction can be entered without the employee's address being known, and conversely the employee's address can be entered before any paychecks and deductions have been entered into the database.

# Cardinal Relationships Between Segments

FOCUS supports most types of cardinal relationships between groups of data, including:

- One-to-one (1:1)
- One-to-many (1:M)
- Many-to-many (M:M)

You can define these relationships between:

- Instances of different segments.
- Instances of the same segment—that is, a recursive or bill-of-materials relationship.
- Segments from the same type of data source.
- Segments from different types of data sources—for example, between a DB2 table and a VSAM record.

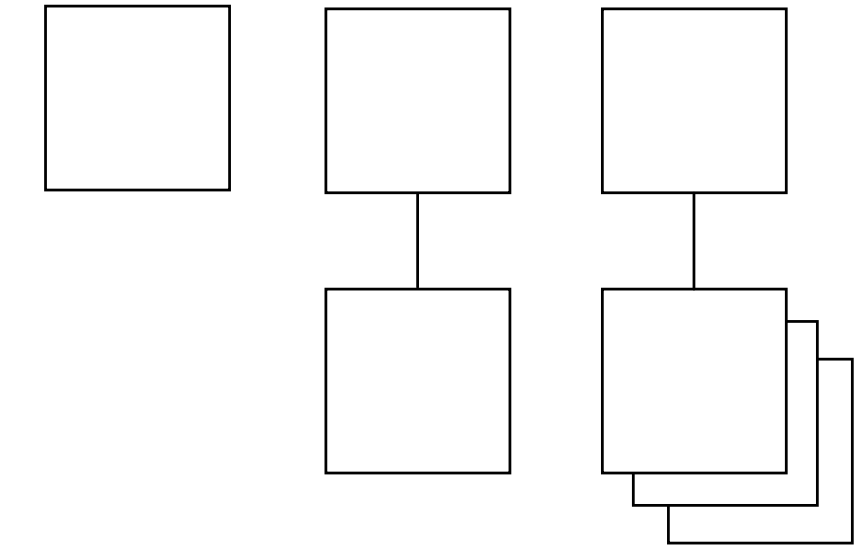
If you are using a network database, you can also “rotate” the database after you have defined it, creating an alternate view that reverses some of the data relationships and enables you to access the segments in a different order.

# One-to-One Relationships

The fields in a segment have a one-to-one relationship with each other. Segments can also exhibit a one-to-one relationship: each instance of a parent segment can be related to one instance of a child segment, as shown in the following diagram. Because it is one-to-one, it will never be related to more than one instance of the child. Of course, not every parent instance needs to have a matching child instance.

FOCUS refers to the child in a one-to-one relationship as a unique segment. The term refers to the fact that there can never be more than a single child instance.

**One-to-One Relationship (1:1)**



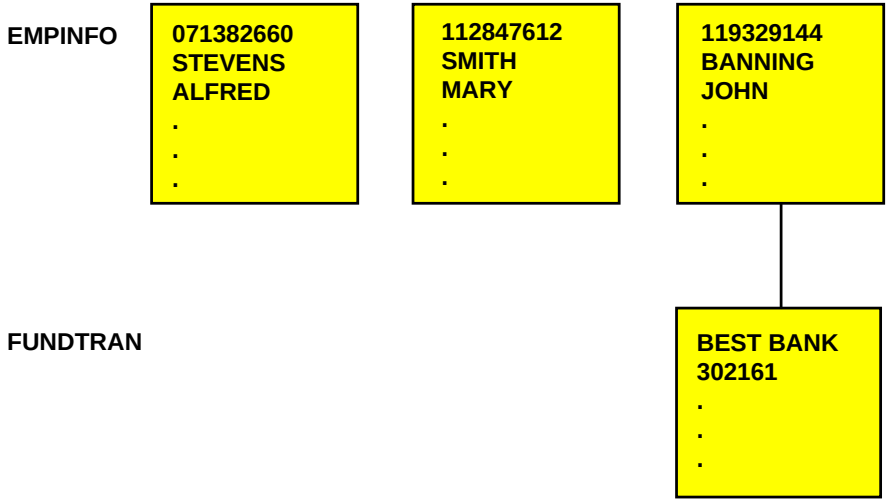
**parent without child is valid** | **parent with one child is valid** | **parent with many children is invalid**

*Example* **Understanding One-to-One Relationships**

Consider the EMPLOYEE database: each EMPINFO segment instance describes one employee's ID number, name, current salary, and other information. Some employees have joined the Direct Deposit program, which deposits their paycheck directly into their bank account each week. For these employees the database also contains the name of their bank and their account number.

Because only one set of bank information is needed for each employee (since each employee's paycheck is deposited into only one account), there is a one-to-one relationship between employee ID fields and bank information fields. But because there is limited participation in the Direct Deposit program, only some employees have bank information; most of the employees do not need the bank fields.

The database was designed with storage efficiency in mind, and so the bank fields have been put into a separate segment called FUNDTRAN; space will be used for only the banking information—that is, an instance of FUNDTRAN will only be created—if it is needed. Compare this to including the banking fields in the parent segment (EMPINFO): the EMPINFO segment for each employee would reserve space for the banking fields, even though they would be empty in most cases.



## Where to Use One-to-One Relationships

You can specify a segment to FOCUS as unique to enforce a one-to-one relationship when users report on data.

When users report from a segment described as unique, the report request treats the unique segment as an extension of its parent. If the unique segment has multiple instances, the report retrieves only one. If the unique segment has no instances, the report substitutes default values for the missing segment's fields: zero (0) for numeric fields, blank ( ) for alphanumeric fields, and the null value for fields that have the MISSING attribute specified. The MISSING attribute is described in Chapter 4, *Describing Individual Fields*.

## Implementing One-to-One Relationships in Relational Databases

You can describe this relationship by joining the tables in the Master File and specifying a SEGTYPE of U for the child table. For more information on joining the tables in a Master File, see the appropriate Interface manual or, in a client/server environment, see the EDA documentation for your server. Alternatively, you can join the tables by issuing the JOIN command without the ALL phrase and turning off the SQL Optimization facility with the SET OPTIMIZATION command.

## Implementing One-to-One Relationships in Sequential Data Files

You can specify this relationship between two records by issuing the JOIN command without the ALL phrase.

## Implementing One-to-One Relationships in FOCUS Databases

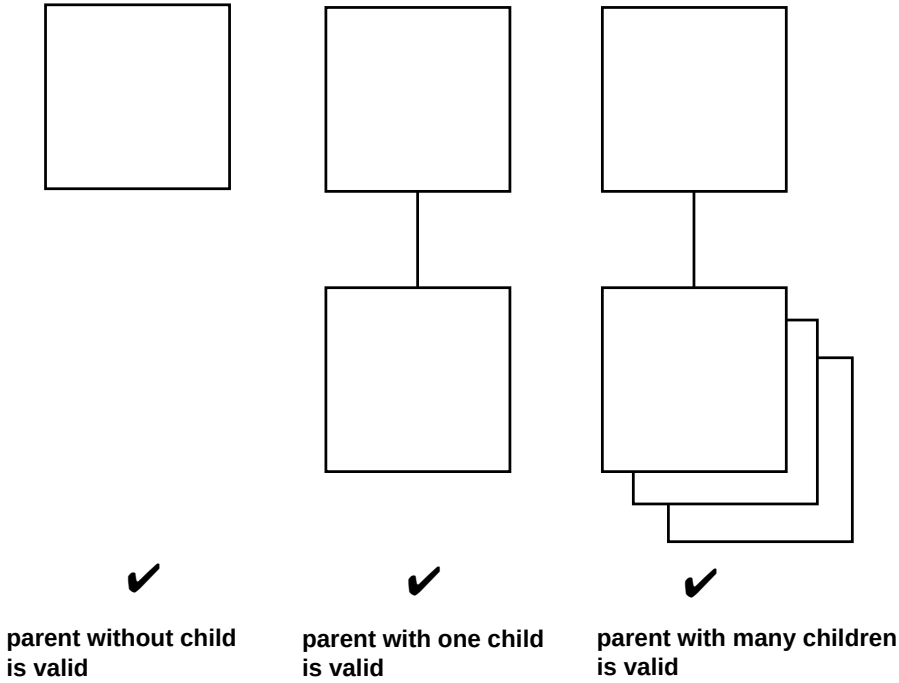
You can describe this relationship by specifying a SEGTYPE of U for the child segment. Alternately, you can join the segments by issuing the JOIN command without the ALL phrase, or by specifying a unique join in the Master File using a SEGTYPE of KU (for a static join) or DKU (for a dynamic join). All of these SEGTYPE values are described in Chapter 7, *Describing FOCUS Databases.*

Of course, you can also describe a one-to-one segment relationship, in the Master File or using the JOIN command, as a one-to-many relationship. This technique gives you greater flexibility but does not enforce the one-to-one relationship when reporting or entering data and does not use resources as efficiently.

# One-to-Many Relationships

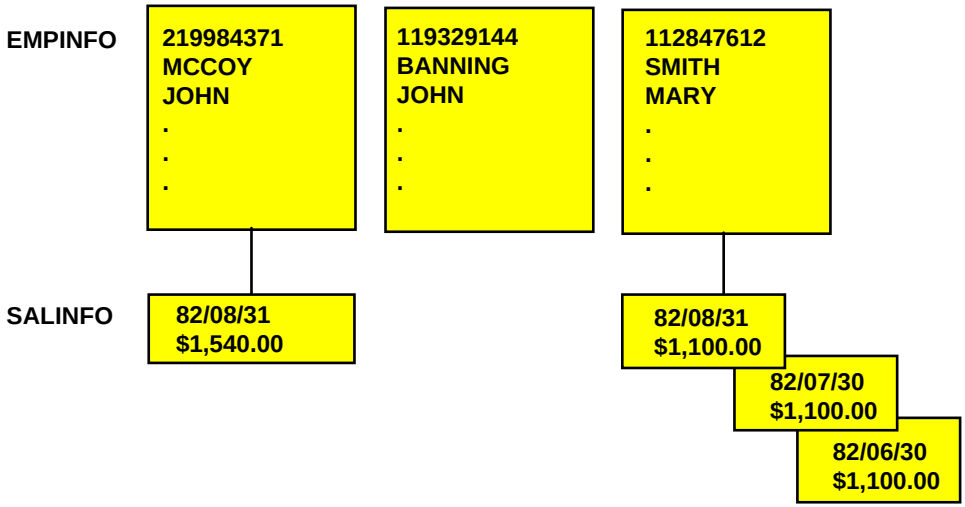
The most common relationship between two segments is the one-to-many relationship: each instance of a parent segment can be related to one or more instances of a child segment, as shown in the following diagram. Of course, not every parent instance needs to have matching child instances.

**One-to-Many Relationship (1:M)**



***Example*** **Understanding One-to-Many Relationships**

Consider the EMPLOYEE database: each EMPINFO segment instance describes one employee's ID number, name, current salary, and other information. Each SALINFO segment contains an employee's gross salary for each month. Most employees work for many months, and so the relationship between EMPINFO and SALINFO is one-to-many.




## Implementing One-to-Many Relationships in Relational Databases

You can describe this relationship by joining the tables in the Master File and specifying a SEGTYPE of S0 for the child table. For more information on joining the tables in a Master File, see the appropriate Interface manual. Alternately, you can join the tables by issuing the JOIN command with the ALL phrase.

## Implementing One-to-Many Relationships in VSAM and Sequential Data Files

You can describe a one-to-many relationship between a record and a group of multiply-occurring fields within the record by using the OCCURS attribute to specify how many times the field (or fields) occur, the POSITION attribute to specify where in the record they occur if they are not at the end of the record, the ORDER field if you wish to determine the sequence number of an occurrence, and the PARENT attribute to indicate the relationship between the singly-occurring and multiply-occurring fields. The OCCURS and POSITION attributes and the ORDER field are all described in Chapter 5, *Describing Sequential Data Files*.

You can describe a one-to-many relationship between different records by using a RECTYPE field to indicate the type of each record, and the PARENT attribute to indicate the relationship between the different records. RECTYPE fields are described in Chapter 5, *Describing Sequential Data Files*, and Chapter 6, *Describing ISAM and VSAM Files*.

You can also specify a one-to-many relationship between two records in different data files by issuing the JOIN command with the ALL phrase or defining the join in the Master File. See the *Creating Reports* manual for information about the JOIN command, and see Chapter 8, *Defining a Join in a Master File*, for information about joins in a Master File.

## Implementing One-to-Many Relationships in FOCUS Databases

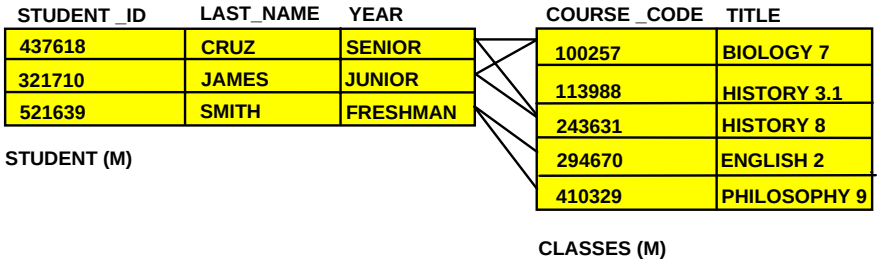
You can describe this relationship by specifying a SEGTYPE of Sn or SHn for the child segment. Alternatively, you can join the segments by issuing the JOIN command with the ALL phrase, or by specifying a join in the Master File with a SEGTYPE of KM (for a static join) or DKM (for a dynamic join). All of these SEGTYPE values are described in Chapter 7, *Describing FOCUS Databases*.

# Many-to-Many Relationships

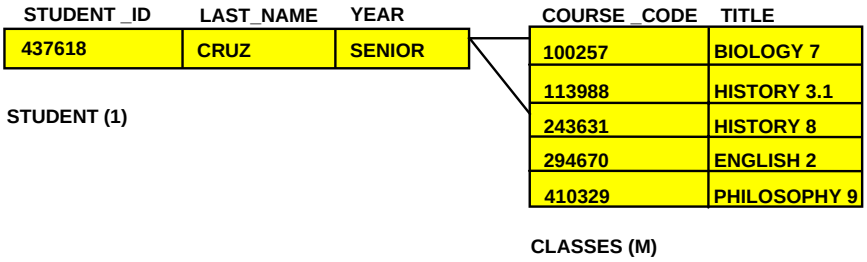
A less commonly used relationship is many-to-many: each instance of one segment can be related to one or more instances of a second segment, and each instance of the second segment can be related to one or more instances of the first segment. It is possible to implement this relationship directly between two relational tables and indirectly between segments of other types of data sources.

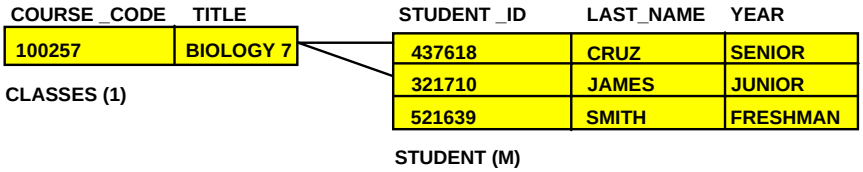
## Implementing Many-to-Many Directly

A direct many-to-many relationship can exist between two relational tables. For example, imagine two tables called STUDENT and CLASSES: the former contains one row for each student enrolled at a college, and the latter contains one row for each class offered at the college. Each student can take many classes, and each class can be taken by many students. This is illustrated in the following diagram:




When the M:M relationship is seen from the perspective of either of the two tables, it looks like a 1:M relationship: one student taking many classes (1:M from the perspective of STUDENT), or one class taken by many students (1:M from the perspective of CLASSES).



When you use FOCUS to report from or update the tables, at any one time FOCUS sees the M:M relationship from the perspective of one of the tables—that is, it sees a 1:M relationship. You decide which table's perspective FOCUS will use by making that table the parent (host) segment, either in the Master File or JOIN command. You describe the join in the Master File or JOIN command as you would for a standard one-to-many relationship.

For example, you could use the JOIN command to describe the relationship from the perspective of the STUDENT table as follows:

```
JOIN STUDENT_ID IN STUDENT TO ALL STUDENT_ID IN CLASSES
```

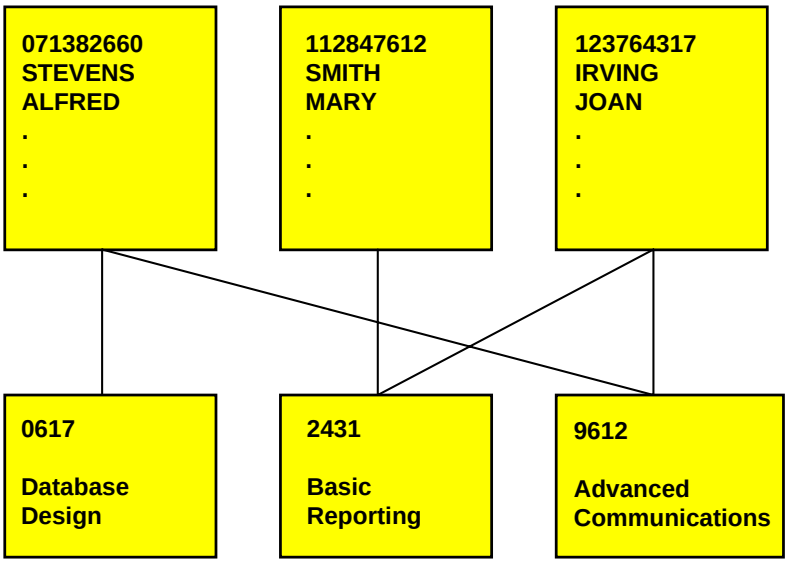
You could describe the relationship from the perspective of the CLASSES table as follows:

```
JOIN CLASS_NUM IN CLASSES TO ALL CLASS_NUM IN STUDENT
```

## Implementing Many-to-Many Indirectly

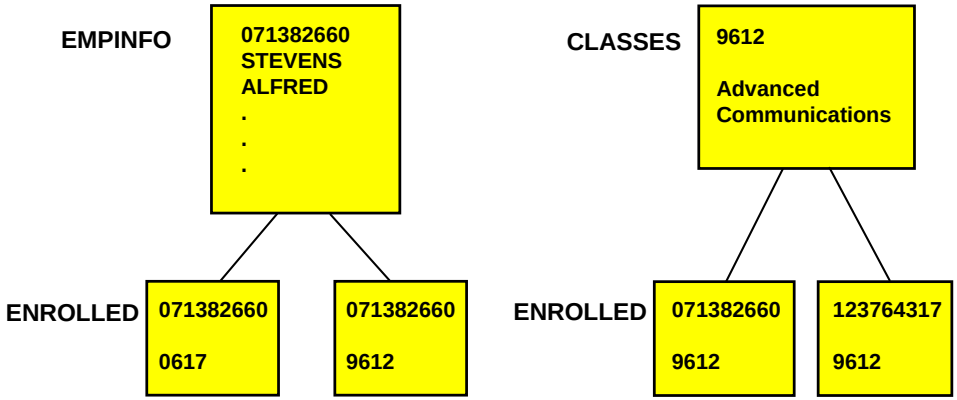
Some non-relational data sources cannot represent a many-to-many relationship directly. However, they can represent it indirectly, and you can describe it as such to FOCUS.

For example, consider the EMPINFO segment in the EMPLOYEE database and the CLASSES segment in a hypothetical SCHOOL database. Each instance of EMPINFO describes one employee, and each instance of CLASSES would describe one course. Each employee can take many courses, and each course can be taken by many employees, so this is a many-to-many relationship.

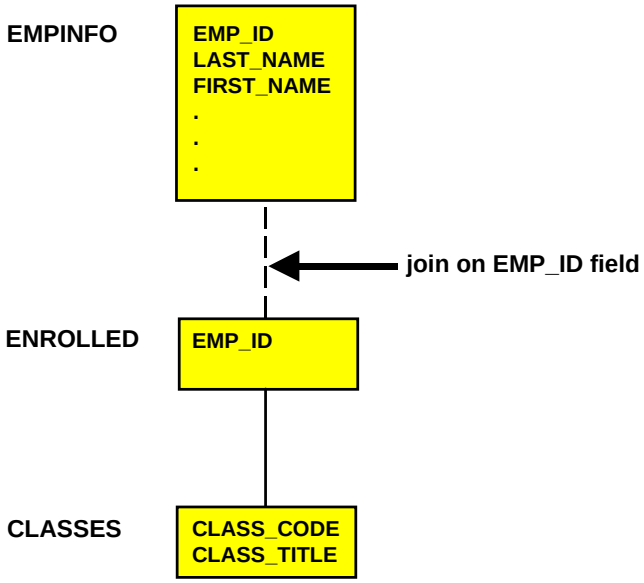


However, because some types of data sources cannot represent such a relationship directly, we need to introduce a mediating segment called ENROLLED. This new segment contains the keys from both of the original segments, EMP_ID and CLASS_CODE, and, in a sense, it represents the relationship between the two original segments. The new segment breaks up the M:M relationship into two 1:M relationships: Each instance of EMPINFO can be related to many instances of ENROLLED (because one employee can be enrolled in many classes), and each instance of CLASSES can be related to many instances of ENROLLED (because one class can have many employees enrolled in it).

This is illustrated in the following diagram.



The next step is to make the mediating segment a child of one of the two original segments. For example, you can design the SCHOOL database so that CLASSES is the root and ENROLLED is the child of CLASSES. Note that when ENROLLED was an “unattached” segment it explicitly contained the keys (EMP_ID and CLASS_CODE) from both original segments; now that we are making it part of the SCHOOL database, CLASS_CODE is implied by the parent-child relationship with CLASSES, and it can be removed from ENROLLED. You can then join EMPINFO and ENROLLED together:



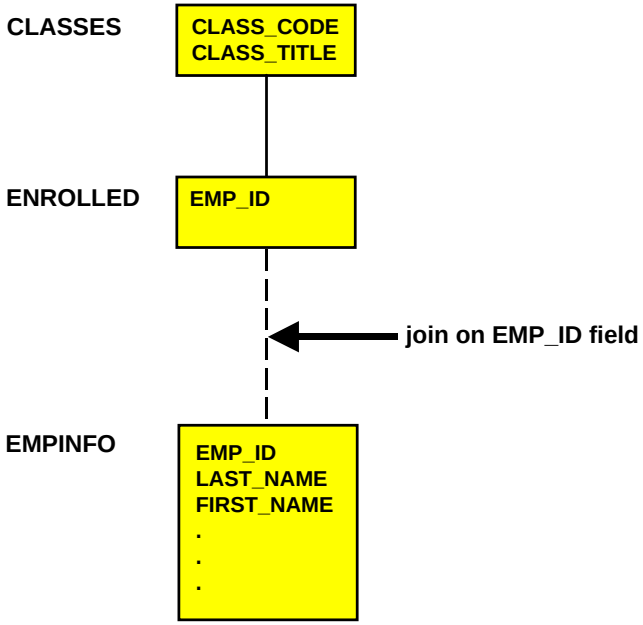

When the original M:M relationship is seen from this perspective, it looks like a 1:M:1 relationship. That is, one employee (EMPINFO) is enrolled many times (ENROLLED), and each enrollment is for a single class (CLASSES).

When you use FOCUS to report from or update the new structure, at any one time FOCUS will see the relationship from the perspective of one of the original segments—in this case, from EMPINFO or CLASSES. You decide which segment’s perspective FOCUS will use by making that segment the parent in the join. You describe the join using the JOIN command (or for FOCUS databases, alternately in the Master File). If you make the mediating segment, in this case ENROLLED, the child (cross-referenced) segment in the join, you implement it as a standard one-to-many relationship; if you make it the parent (host), you implement it as a standard one-to-one join.

For example, you could use the JOIN command to describe the relationship from the perspective of the CLASSES segment—that is, making ENROLLED the join's host—as follows:

```
JOIN EMP_ID IN ENROLLED TO EMP_ID IN EMPINFO
```

The new structure would look like this:




Another example that uses a join defined in the Master File is illustrated by the sample FOCUS databases EMPLOYEE and EDUCFILE. Here, ATTNDSEG is the mediating segment between EMPINFO and COURSEG.

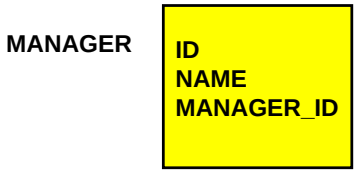
# Recursive Relationships

Generally, you use one-to-one and one-to-many relationships to join two different segments, usually in two different data sources. However, you can also join the *same data source* to itself, and even the same segment to itself. This technique, which has many useful applications, is called a recursive join.

Recursive joins are described in more detail in the *Creating Reports* manual.

## Recursive Joins With a Single Segment

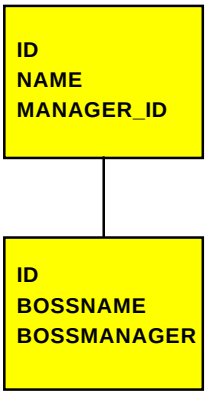
For example, assume that you have a single-segment data file called MANAGER, which includes the ID number of an employee, the employee's name, and the ID number of the employee's manager:




If you want to generate a report showing every employee's ID number, name, and manager's ID number and name, you would need to join the segment to itself. You could issue the following command

```
JOIN MANAGER_ID IN MANAGER TO ID IN MANAGER AS BOSS
```

which would create the following structure:



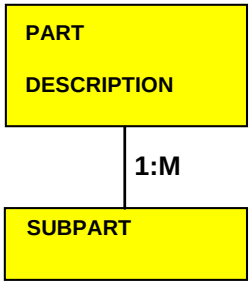
Note that you can uniquely refer to fields in cross-referenced segments by prefixing them with the first four letters of the join name (BOSS in this example). The only exception is the cross-referenced field, for which the alias is prefixed instead of the field name.

Once you have issued the join, you would be able to use FOCUS to generate a report like the following:

```
PAGE 1

ID          NAME           MANAGER_ID      BOSSNAME
--          ----           ----------      --------
026255      JONES          837172          CRUZ
308743      MILBERG        619426          WINOKUR
846721      YUTANG         294857          CAPRISTI
743891      LUSTIG         089413          SMITH
585693      CAPRA          842918          JOHNSON
```

## Recursive Joins With Multiple Segments

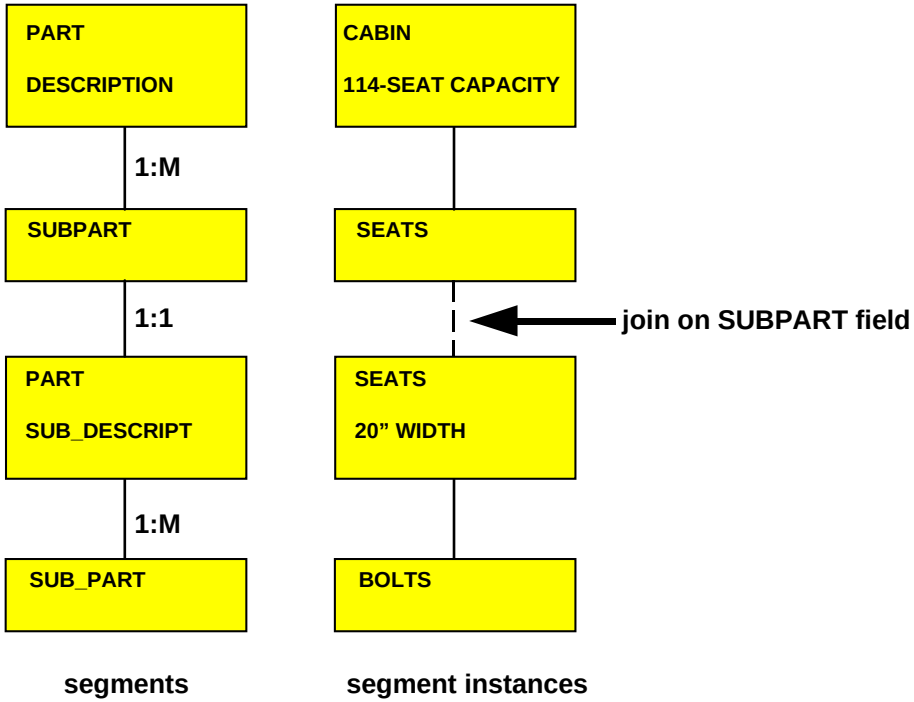
You can recursively join larger structures as well. For example, imagine a two-segment database called AIRCRAFT that stores a bill-of-materials for an aircraft company. The root segment has the name and description of a part, and the child segment has the name of a subpart. For each part, there can be many subparts.



While many of the larger parts are constructed of several levels of subparts, some of these subparts, such as bolts, are used throughout aircraft at many different levels; giving each occurrence of a subpart its own segment instance would produce much redundancy. Instead, we can use the two-segment design shown previously and then join the database to itself:

```
JOIN SUBPART IN AIRCRAFT TO PART IN AIRCRAFT AS SUB_PART
```

This produces the following data structure:

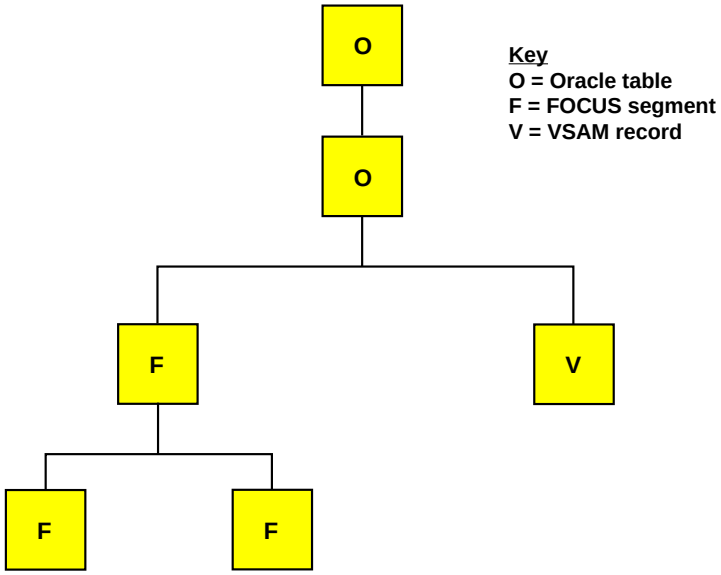


# Relating Segments From Different Types of Data Sources

The JOIN command enables you to join segments from different types of data sources, creating temporary data structures that contain related information from otherwise incompatible sources. For example, you could join two Oracle tables to a FOCUS database to a VSAM data file, as illustrated in the following diagram.

Joins between VSAM and fixed-format data files are also supported in a Master File, as described in Chapter 8, *Defining a Join in a Master File.*

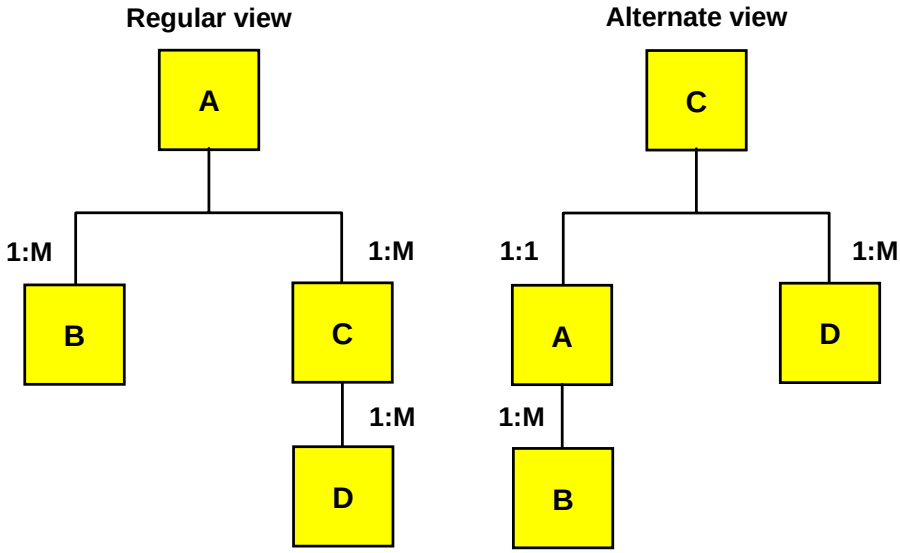
For detailed information on using the JOIN command with different types of data sources, see the *Creating Reports* manual.



# Rotating a Database: Alternate Views

If you are using a network database or certain hierarchical data sources such as FOCUS, you can “rotate” the database after you have described it, creating an alternate view that changes some of the segment relationships and enables you to access the segments in a different order. By customizing the access path in this way, you can enable it to be accessed faster for a given application.

For example:




You can even join hierarchical and/or network databases together and then create an alternate view of the joined structure, selecting the new root segment from the host database.

Using an alternate view can be very helpful when you want to generate a report using record-selection criteria based on fields found in a lower segment (such as segment C in the previous diagram). You could report from an alternate view that makes this the root segment; FOCUS will begin its record selection based on the relevant segment, and avoid reading irrelevant ancestral segments.

When you report from a FOCUS database using an alternate view, FOCUS will access the data even faster if both of the following conditions are satisfied:

- The field on which the alternate view is based is indexed—that is, the INDEX attribute in the Master File is set to I.
- You use the field in a record selection test, using the WHERE or IF phrases, and make the selection criteria an equality or range test.

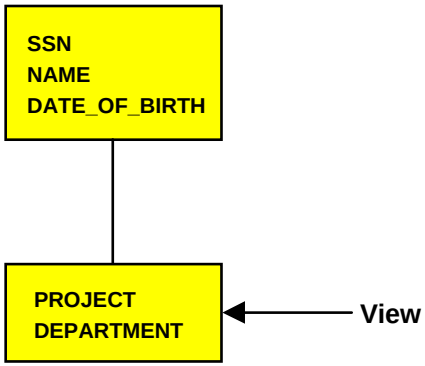
An alternate file view can be requested on any segment in a file, except a cross-referenced segment. You request an alternate view with the TABLE command by naming a field from the segment you wish to view as the new “root” segment. This field may not be a defined field. The only restriction on requesting an alternate file view is that the field on which it is requested must be a real field in the file. It cannot reside on a cross-referenced segment or be a defined field.

## Other Uses of an Alternate File View

The alternate view capability can be used effectively by the file designer in many situations. Some ideas are:

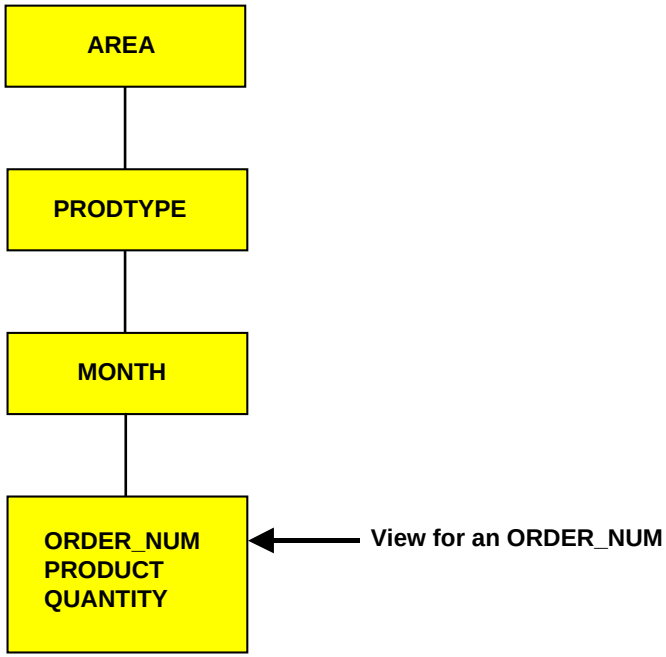
- Files with active individual record maintenance: The files can be structured for efficient update and management. For example, primary record keys can be placed at the top of the file, even though requests frequently screen on other fields. The hierarchy does not assist in record selection, so views are used.

  Consider a personnel system where the employee identity (SSN) is in the root segment and the department number of the project an employee is working on is in a descendant segment. Access to all employees in a given department is obtained using the view from the department.




  Note that a given SSN can also be retrieved faster by using a view if the SSN values are indexed.

- Individual record identified in descendant segments: You can use an alternate file view to access a detail segment that is deep in the hierarchy. Consider a sales analysis situation. The file has ample structure for AREA, PRODUCT TYPE, and MONTH, but a request for a particular ORDER_NUM is easily handled by a view.




  Note that in this view, ORDER_NUM is unique.

- Many missing instances of data: When a particular segment is often not available, then screening on it means that fewer segments have to be examined. For instance, if there are 10,000 occurrences of a parent segment, but only 2,000 of these have a given child segment, it is faster to view the file from the vantage point of the 2,000 when the given child is involved in the screening in the request.

For more information about using alternate views in report requests, see the *Creating Reports* manual.

## *Syntax* How to Specify an Alternate View

To specify an alternate view, simply append a field name to the file name in the reporting command, using the syntax

*filename.fieldname*

where:

*filename*
Is the name of the data file on which you are defining the alternate view.

*fieldname*
Is a field located in the segment that you are defining as the alternate root. The field must be a real field, not a temporary field defined with the DEFINE attribute or the DEFINE or COMPUTE commands.

If the field is declared in the Master File with the FIELDTYPE attribute set to I, and you use the alternate view in a report, you must use the field in an equality selection or range test.

## *Example* Specifying an Alternative View

For example, if you want to report from the EMPLOYEE database using an alternate view that makes the DEDUCT segment an alternate root, you could issue the following TABLE FILE command:

```
TABLE FILE EMPLOYEE.DED_CODE
```

# 4 Describing Individual Fields

**Topics:**

- The Field's Name: FIELDNAME
- The Field's Synonym: ALIAS
- The Stored Data Type: ACTUAL
- The Displayed Data Type: FORMAT
- Sliding Window Field Attributes for Handling Two-Digit Years
- Null or MISSING Values: MISSING
- Describing Temporary Fields: DEFINE
- Validating Data: ACCEPT
- Online Help Information: HELPMESSAGE
- Alternative Report Column Titles: TITLE
- Documenting the Field: DESCRIPTION

A field is the smallest meaningful element of data in a data source, but it can exhibit a number of complex characteristics. FOCUS uses Master File attributes to describe these characteristics:

- The name of the field recognized by FOCUS and described by the FIELDNAME attribute.
- Another name for the field—either its original name as defined to its native data management system, or (for some types of data sources) a synonym of your own choosing, or (in some special cases) a pre-defined value that tells FOCUS how to interpret the field—that you can use as an alternative name in FOCUS requests. This alternative name is defined by the ALIAS attribute.

- How the field stores and displays data, specified by the ACTUAL, FORMAT, and MISSING attributes.

  The ACTUAL attribute describes the type and length of the data as it is actually stored in the data source. For example, a field might be alphanumeric and 15 characters in length. Note that FOCUS databases do not use the ACTUAL attribute, and instead use FORMAT to describe the data both as it is stored in the file and as it is formatted by FOCUS, since these are identical.

  The FORMAT attribute, which is also known by its alias, USAGE, describes how you want a field to be formatted when FOCUS displays it in reports. You can also specify edit options such as date formats, floating dollar signs, and zero suppression.

  The MISSING attribute enables null values to be entered into and read from a field in data sources that support null data, such as FOCUS databases and most relational data sources.

- The option that a field be temporary—that is, not stored in the data source—and have its value derived from information already in the data source. Temporary fields are specified by the DEFINE attribute.
- Optional field documentation for the developer, contained in the DESCRIPTION attribute.
- Acceptable data-entry values for the field, specified by the ACCEPT attribute.
- Online help information about the field that an end user can display during an application, as described by the HELPMESSAGE attribute.
- An alternative report column title for the field, described by the TITLE attribute.
- A 100-year window that assigns a century value to a two-digit year stored in the field. Two attributes define this window, DEFCENT and YRTHRESH.

# The Field's Name: FIELDNAME

You identify a field to FOCUS using FIELDNAME, the first attribute specified in a field declaration in the Master File. You can assign any name to a field, regardless of its name in its native data source. Likewise, for FOCUS databases, you can assign any name to a field in a new database. The name of a field should be unique within the data source.

Your reporting applications may influence your choice of field name: when you generate a report, each column title in the report defaults to the name of the field displayed in that column, so it will help report readers if you assign meaningful field names. Of course, you do not need to rely upon this default: you can specify a different column title within a given report by using the AS phrase in that report request—as described in the *Creating Reports* manual—or a different default column title for all reports by using the TITLE attribute in the Master File, as described in *Alternative Report Column Titles: TITLE* on page 4-44.

## *Syntax* How to Identify the Field Name

The syntax of the FIELDNAME attribute is

```
FIELD[NAME] = field_name
```

where:

*field_name*

Is the name you want to use to identify this field to FOCUS. It can be up to 66 characters long. Some restrictions apply to names longer than 12 characters, as described below. The name 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 not use field names of the type Cn, En, and Xn (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.

If you need to use special characters because of a field's report column title, consider using the TITLE attribute in the Master File to specify the title, as described in *Alternative Report Column Titles: TITLE* on page 4-44.

## *Reference* Usage Notes for FIELDNAME

Note the following rules when using FIELDNAME:

- **Alias.** FIELDNAME has an alias of FIELD.
- **Changes.** In a FOCUS database, if the INDEX attribute has been set to I—that is, if a FOCUS index has been created for the field—you cannot change the field name. In all other situations you can change the field name.

### *Reference* Restrictions for FIELDNAME

The following restrictions apply to field names and aliases longer than 12 characters (that is, long names):

- Joins

  You cannot use a long name to specify a join:

  - In a JOIN command, you cannot use it for the host and cross-referenced fields.
  - In a multi-table Master File for a relational data source, you cannot use it for the KEYFLD and IXFLD attributes in the Access File.

- Indexed fields and text files in FOCUS databases cannot have field names longer than 12 characters. They can have long ALIAS names.
- The SQL Translator supports field names up to 48 characters.
- A field name specified in an alternate file view cannot be long or qualified.
- CHECK FILE

  The CHECK FILE command's PICTURE and HOLD options display the first 11 characters of long names within the resulting diagram or HOLD file. A caret (>) in the 12th position indicates that the name is longer than the displayed portion.

- ?FF, ?HOLD, ?DEFINE

  These display up to 31 characters of the name and display a caret (>) in the 32nd character to indicate a longer field name.

## Using Long and Qualified Field Names

In Master Files, field names and aliases can have a maximum length of 66 characters. However, before defining a field name longer than 48 characters, you must consider how the name will be referenced in requests.

Requests can qualify all referenced field names and aliases with file and/or segment names. This technique is useful when duplicate field names exist across segments in a Master File or in files that are joined. But, although the qualifiers and qualification characters are valid only in requests, not in Master Files, the 66-character maximum includes any qualifiers and qualification characters used with the field name in requests. Therefore, if you define a 66-character name in the Master File, you cannot use qualifiers with the name in a request.

The maximum of 66 characters includes the name of the field or alias, plus an eight-character maximum for each field qualifier (Master File name and segment name), plus a qualification character (usually a period) for each qualifier. You may use a unique truncation of a 66-character name with a qualifier.

Defined and computed field names may also be up to 66 characters. Text fields and indexed fields in FOCUS Master Files are limited to 12 characters; however, the aliases for text and indexed fields may be up to 66 characters. Field names up to 66 characters are displayed as column titles in TABLE reports if there is no TITLE attribute or AS phrase.

The fully qualified name of the field EMP_ID in the EMPINFO segment of the EMPLOYEE file is:

```
EMPLOYEE.EMPINFO.EMP_ID
```

The default value for the SET FIELDNAME command, SET FIELDNAME=NEW, activates long and qualified field names. The syntax is described in the *Developing Applications* manual.

**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. For more information about this amper variable, see the *Developing Applications* manual.

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

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

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

## *Syntax* How to Specify Qualified Field Names in Requests

Specify qualified field names in requests with the following syntax

[*filename.*][*segname.*]*fieldname*

where:

*filename*
: Is the one- to eight-character name of the Master File or tag name. Tag names are used with the JOIN and COMBINE commands.

*segname*
: Is the one- to eight-character name of the segment in which the field resides.

*fieldname*
: Is the name of the field.

### *Syntax* How to Change the Qualifying Character

You can change the qualifying character with the command:

```
SET QUALCHAR = qualcharacter
```

The period (.) is the default qualifying character. For further information about the SET QUALCHAR command and valid qualifying characters, see the *Developing Applications* manual.

## Using Duplicate Field Names

Field names are considered duplicates when you can reference two or more fields with the same field name or alias. Duplication may occur:

- If a name appears multiple times within a Master File.
- In a JOIN between two or more Master Files, or in a recursive JOIN.
- If you issue a COMBINE and do not specify a prefix.

Duplicate fields (those having the same field name and alias) are not allowed in the same segment. FOCUS never accesses the second occurrence, and the following warning message is generated when you issue CHECK and CREATE FILE:

```
(FOC1829) WARNING. FIELDNAME IS NOT UNIQUE WITHIN A SEGMENT: fieldname
```

Duplicate field names may exist across segments in a Master File. To retrieve such a field, you must qualify its name with the segment name in a request. If a field that appears multiple times in a Master File is not qualified in a request, the first field encountered in the Master File is retrieved.

**Note:** If a Master File includes duplicate field names for real fields and/or DEFINE fields, FOCUS uses the following logic when retrieving a field:

- If only DEFINE fields are duplicated, the last DEFINE field is retrieved.
- If only real fields are duplicated, the first real field is retrieved.
- If a Master File has both a real field and one or more DEFINE fields with the same name, the last DEFINE field is retrieved.
- If a field defined outside of a Master File has the same name as a DEFINE or real field in a Master File, the last field defined outside of the Master File is retrieved.

Reports can include qualified names as column titles. The SET QUALTITLES command, discussed in the *Developing Applications* manual, determines whether reports display qualified column titles for duplicate field names. With SET QUALTITLES=ON, reports display qualified column titles for duplicate field names even when the request itself does not specify qualified names. You can issue the SET QUALTITLES command from the FOCUS command level or in a request. The default value, OFF, disables qualified column titles.

## Rules for Evaluating Qualified Field Names

FOCUS uses the following rules to evaluate qualified field names:

- The maximum field name qualification is file name.segname.field name. For example:

  ```
  TABLE FILE EMPLOYEE
  PRINT EMPLOYEE.EMPINFO.EMP_ID
  END
  ```

  includes EMP_ID as a fully qualified field. The file name, EMPLOYEE, and the segment name, EMPINFO, are the field qualifiers.

  Qualifier names can also be duplicated. For example:

  ```
  FILENAME=CAR, SUFFIX=FOC
      SEGNAME=ORIGIN, SEGTYPE=S1
          FIELDNAME=COUNTRY, COUNTRY, A10, $
      SEGNAME=COMP, SEGTYPE=S1, PARENT=ORIGIN
          FIELDNAME=CAR, CARS, A16, $
              .
              .
              .

  TABLE FILE CAR
  PRINT CAR.COMP.CAR
  END
  ```

  This request prints the field with alias CARS. Both the file name and field name are CAR.

- A field name can be qualified with a single qualifier, either its file name or its segment name. For example:

  ```
  FILENAME=CAR, SUFFIX=FOC
      SEGNAME=ORIGIN, SEGTYPE=S1
          FIELDNAME=COUNTRY, COUNTRY, A10, $
      SEGNAME=COMP, SEGTYPE=S1, PARENT=ORIGIN
          FIELDNAME=CAR, CARS, A16, $
              .
              .
              .

  TABLE FILE CAR
  PRINT COMP.CAR AND CAR.CAR
  END
  ```

  This request prints the field with alias CARS twice.

  When there is a single qualifier, segment name takes precedence over file name. Therefore, if the file name and segment name are the same, the field qualified by the segment name is retrieved.

- If a field name begins with characters that are the same as the name of a prefix operator, it may be unclear whether a request is referencing that field name or a second field name prefixed with the operator; FOCUS retrieves the value of the first field, not the value calculated by applying the prefix operator to the second field. In the next example, there is a field whose unqualified field name is CNT.COUNTRY and another whose field name is COUNTRY:

  ```
  FILENAME=CAR, SUFFIX=FOC
      SEGNAME=ORIGIN, SEGTYPE=S1
          FIELDNAME=CNT.COUNTRY, ACNTRY, A10, $
          FIELDNAME=COUNTRY, BCNTRY, A10, $

  TABLE FILE CAR
  SUM CNT.COUNTRY
  END
  ```

  In this request, the string “CNT.COUNTRY” is interpreted as a reference to the field named CNT.COUNTRY, not as a reference to the prefix operator CNT. applied to the field named COUNTRY; therefore, the request sums the field whose alias is ACNTRY. Although the field name CNT.COUNTRY contains a period as one of its characters, it is an unqualified field name; it is not a qualified name or a prefix operator acting on a field name, neither of which is allowed in a Master File. The request does not count instances of the field whose alias is BCNTRY.

- If a Master File has either a file name or segment name that is the same as a prefix operator, the value of the field within the segment is retrieved in requests, not the value calculated by applying the prefix operator to the field. For example:

  ```
  FILENAME=CAR, SUFFIX=FOC
      SEGNAME=ORIGIN, SEGTYPE=S1
          FIELDNAME=COUNTRY, COUNTRY, A10, $
      SEGNAME=PCT, SEGTYPE=S1, PARENT=ORIGIN
          FIELDNAME=CAR, CARS, I2, $

  TABLE FILE CAR
  SUM PCT.CAR PCT.PCT.CAR
  BY COUNTRY
  END
  ```

  This request sums the field with alias CARS first and then the percent of CARS by COUNTRY.

- When a qualified field name can be evaluated as a choice between two levels of qualification, the field name with the higher level of qualification takes precedence.

  In the following example, the choice is between an unqualified field name (the field named ORIGIN.COUNTRY in the ORIGIN segment) and a field name with segment name qualification (the field named COUNTRY in the ORIGIN segment). The field with segment name qualification is retrieved:

  ```
  FILENAME=CAR, SUFFIX=FOC
      SEGNAME=ORIGIN, SEGTYPE=S1
          FIELDNAME=ORIGIN.COUNTRY, OCNTRY, A10, $
          FIELDNAME=COUNTRY, CNTRY, A10, $

  TABLE FILE CAR
  PRINT ORIGIN.COUNTRY
  END
  ```

  This request prints the field with alias CNTRY. To retrieve the field with alias OCNTRY, qualify its field name, ORIGIN.COUNTRY, with its segment name, ORIGIN:

  ```
  PRINT ORIGIN.ORIGIN.COUNTRY
  ```

- When a qualified field name can be evaluated as a choice between two field names with the same level of qualification, the field with the shortest basic field name length is retrieved. For example:

  ```
  FILENAME=CAR, SUFFIX=FOC
      SEGNAME=CAR, SEGTYPE=S1
          FIELDNAME=CAR.CAR, CAR1, A10, $
      SEGNAME=CAR.CAR, SEGTYPE=S1, PARENT=CAR
          FIELDNAME=CAR, CAR2, A10, $

  TABLE FILE CAR
  PRINT CAR.CAR.CAR
  END
  ```

  In this example, it is unclear if you intend CAR.CAR.CAR to refer to the field named CAR.CAR in the CAR segment or the field named CAR in the CAR.CAR segment. (In either case, the name CAR.CAR is an unqualified name that contains a period, not a qualified name. Qualified names are not permitted in Master Files.)

  No matter what the intention, the qualified field name is exactly the same and there is no obvious choice between levels of qualification.

  Since the field with alias CAR2 has the shortest basic field name length, CAR2 is printed. This is different from the prior example where the choice was between two levels of qualification. To retrieve the CAR1 field, you must specify its alias.

# The Field's Synonym: ALIAS

You can assign every field an alternative name, or alias. A field's alias may be its original name as defined to its native data source, any name of your choosing, or in special cases, a pre-defined value. The way in which you assign the alias is determined by the type of data source and, in special cases, the role the field plays in the file. Once it has been assigned, you can use this alias in FOCUS requests as a synonym for the regular field name. You assign this alternative name using the ALIAS attribute.

*Example* **Using a Field Synonym**

For example, in the EMPLOYEE database, the name CURR_SAL is assigned to a field using the FIELDNAME attribute, and the alternative name CSAL is assigned to the same field using the ALIAS attribute:

```
FIELDNAME = CURR_SAL, ALIAS = CSAL, FORMAT = D12.2M,  $
```

Both names are equally valid within a FOCUS request. The following TABLE requests illustrate this—they are functionally identical, refer to the same field, and produce the same result:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY EMP_ID
END

TABLE FILE EMPLOYEE
PRINT CSAL BY EMP_ID
END
```

## Implementing Field Synonyms

The value you assign to ALIAS must conform to the same naming conventions to which the FIELDNAME attribute is subject, unless stated otherwise. You assign a value to ALIAS in the following way for the following types of data sources:

- **Relational data sources.** ALIAS describes the field's original column name as defined in the relational table.
- **Sequential data files.** ALIAS describes a synonym, or alternative name, that you can use in a FOCUS request to identify the field. You can assign any name as the alias; many users choose a shorter version of the field's primary name—for example, if the field name is LAST_NAME, the alias might be LN. ALIAS is optional.

  Note that ALIAS is used in a different way for sequenced repeating fields, where its value is ORDER, as well as for RECTYPE fields when the data source includes multiple record types. See Chapter 5, *Describing Sequential Data Files*, for more information about using ALIAS.
- **FOCUS databases.** ALIAS describes a synonym, or alternative name, that you can use in a FOCUS request to identify the field. You can assign any name as the alias; many users choose a shorter version of the field's primary name—for example, if the field name is LAST_NAME, the alias might be LN. ALIAS is optional. See Chapter 7, *Describing FOCUS Databases*, for more information about using ALIAS. Aliases can be changed without rebuilding the file. If an alias is referred to in other files, similar changes may be needed in those Master Files.

# The Stored Data Type: ACTUAL

ACTUAL describes the type and length of data as it is actually stored in the data source. While some data types, such as alphanumeric, are universal, others differ between different types of data sources. Some data sources support unique data types. For this reason, the values you can assign to the ACTUAL attribute differ for each type of data source.

### The ACTUAL Attribute

This attribute describes the type and length of your data as it actually exists in the file. The source of this information is your existing description of the file (such as a COBOL FD statement). The ACTUAL attribute is one of the distinguishing characteristics of a Master File for external data sources. Since this attribute exists only to describe the format of an external data structure, it is not used in the Master File of a FOCUS data structure.

The syntax for this attribute is

```
ACTUAL = format
```

where *format* consists of codes taken from the following tables.

The following table shows the codes for the types of data FOCUS can read:

| ACTUAL Type | Meaning |
|---|---|
| `DATE` | Four-byte integer internal format, representing the difference between the date to be entered and the date format base date. |
| `A`*n* | Where *n* = 1-256. Alphanumeric characters A-Z, 0-9, and the special characters in the EBCDIC display mode. |
| `D8` | Double-precision, floating-point numbers, stored internally in eight bytes. |
| `F4` | Single-precision, floating-point numbers, stored internally in four bytes. |
| `I`*n* | Binary integers:<br>I1 = single-byte binary integer.<br>I2 = half-word binary integer (2 bytes).<br>I4 = full-word binary integer (4 bytes). |
| `P`*n* | Where *n* = 1-16. Packed decimal internal format. *n* is number of bytes, each of which contains two digits, except for the last byte which contains a digit and the sign (+ or -). For example, P6 means 11 digits plus a sign. |
| `Z`*n* | Where *n* = 1-31. Zoned decimal internal format. *n* is the number of digits, each of which takes a byte of storage. The last digit contains a digit and the sign.<br>If the field contains an assumed decimal point, represent the field with an ACTUAL format of Zn and a USAGE format of Pm.d, where *m* is the total number of digits in the display plus the assumed decimal point, and *d* is the number of decimal places. *m* must be at least 1 greater than the value of *n*. For example, a field with ACTUAL=Z5 and one decimal place would need USAGE=P6.1 (or P7.1, or greater). |

**Note:**

- Unless your file was created by a program, all of the characters will be characters of either type A (alphanumeric) or type Z (zoned decimal).
- Data formats can be used with packed numbers declared with an ACTUAL of less than 8.
- The ASQ. prefix is not valid for a packed field of any length.

FOCUS permits the following conversions from ACTUAL format to USAGE (display) format:

| ACTUAL | USAGE |
|---|---|
| A | A, D, F, I, P, date format |
| D | D |
| DATE | date format |
| F | F |
| I | I, date format |
| P | P, date format |
| Z | D, F, I, P |

The following table shows the FOCUS USAGE and ACTUAL formats for COBOL, FORTRAN, PL1, and Assembler field descriptions.

| COBOL USAGE FORMAT | BYTES OF COBOL PICTURE | FOCUS INTERNAL STORAGE | FOCUS ACTUAL FORMAT | USAGE FORMAT |
|---|---|---|---|---|
| DISPLAY | X(4) | 4 | A4 | A4 |
| DISPLAY | S99 | 2 | Z2 | P3 |
| DISPLAY | 9(5)V9 | 6 | Z6.1 | P8.1 |
| DISPLAY | 99 | 2 | A2 | A2 |
| COMP | S9 | 4 | I2 | I1 |
| COMP | S9(4) | 4 | I2 | I4 |
| COMP* | S9(5) | 4 | I4 | I5 |
| COMP | S9(9) | 4 | I4 | I9 |
| COMP-1** | — | 4 | F4 | F6 |
| COMP-2*** | — | 8 | D8 | D15 |
| COMP-3 | 9 | 8 | P1 | P1 |
| COMP-3 | S9V99 | 8 | P2 | P5.2 |
| COMP-3 | 9(4)V9(3) | 8 | P4 | P8.3 |
| FIXED BINARY(7) (COMP-4) | B or XL1 | 8 | I4 | I7 |

*Equivalent to INTEGER in FORTRAN, FIXED BINARY(31) in PL/1, and F in Assembler.

**Equivalent to REAL in FORTRAN, FLOAT(6) in PL/1, and E in Assembler.

***Equivalent to DOUBLE PRECISION or REAL*8 in FORTRAN, FLOAT(16) in PL/1, and D in Assembler.

**Note:**

1. The USAGE lengths shown are minimum values. They may be larger if desired. Additional edit options may also be added.
2. In USAGE formats, an extra character position is required for the minus sign if negative values are expected.
3. PICTURE clauses are not permitted for internal floating-point items.
4. USAGE length should allow for maximum possible number of digits.
5. In USAGE formats, an extra character position is required for the decimal point.

For information about using ACTUAL with sequential data files, see Chapter 5, *Describing Sequential Data Files.* For information about using ACTUAL with ISAM or VSAM files, see Chapter 6, *Describing ISAM and VSAM Files.* For other types of data sources, see the appropriate Interface manual or, in a client/server environment see the EDA documentation for your server. Note that FOCUS databases do not use the ACTUAL attribute, and instead rely upon the FORMAT attribute to specify both how a field is stored and how it is formatted.

# The Displayed Data Type: FORMAT

This attribute, which is also known as USAGE, describes how you want a field to be formatted when FOCUS displays it in reports or uses it in calculations.

For FOCUS databases, which do not use the ACTUAL attribute, FORMAT also specifies how the field is to be stored. For other types of data sources, you will usually want to assign a FORMAT value that corresponds to the ACTUAL value, to tell FOCUS to format the field as the same data type used to store it in the data source. For instructions on which ACTUAL values correspond to which FORMAT values, see the FOCUS Interface documentation for your data source or, in a client/server environment, see the EDA documentation for your server. For sequential data files, see Chapter 5, *Describing Sequential Data Files*. For ISAM and VSAM files, see Chapter 6, *Describing ISAM and VSAM Files*.

In addition to selecting the data type and length, you can also specify display options such as date formatting, floating dollar signs, and zero suppression. You can use these options to customize how the field is displayed in reports.

*Syntax*

## How to Specify a Data Type Format

The general syntax of the FORMAT attribute is:

```
FORMAT = tl[d]
```

where:

*t*

Is the data type. Valid values are A (alphanumeric), D (floating-point double-precision), F (floating-point single-precision), I (integer), P (packed decimal), D, W, M, Q, or Y used in a valid combination (date), and TX (text).

*l*

Is a length specification. Different data types have different length specifications; see the section for each data type for more information. Note that you do not specify a length for date format fields.

*d*

Is one or more display options. Different data types offer different display options; see the section for each data type for more information.

The complete FORMAT value cannot exceed eight characters.

The values that you specify for type and field length determine the number of print positions that FOCUS allocates for displaying or storing the field. Display options only affect displayed or printed fields. They are not active for non-display retrievals, such as extract files.

Examples and additional information about each format type are provided in the section for that type.

*Reference*

## Usage Notes for FORMAT

Note the following rules when using FORMAT:

- **Alias.** FORMAT has an alias of USAGE.
- **Changes.** For most data sources, you can change the type and length specifications of FORMAT only to other types and lengths valid for that field's ACTUAL attribute. You can change display options at any time.

  For FOCUS databases, you cannot change the type specification. You can change the length specification for I, F, D, and P fields, because this affects only display, not storage. You cannot change the decimal part of the length specification for P fields. You can change the length specification of A (alphanumeric) fields only if you use the REBUILD facility. You can change display options at any time.

# Data Type Formats

You can specify several types of formats:

- **Numeric.** There are four types of numeric formats: integer, floating-point single-precision, floating-point double-precision, and packed decimal.
- **Alphanumeric.**
- **Date.** The date format enables you to define date components such as year, quarter, month, day, and day of week; to sort by date; to do date comparisons and arithmetic with dates; and to automatically validate dates in transactions. Note that for some applications, such as assigning a date value using the DECODE function, you may wish to instead use alphanumeric, integer, or packed-decimal fields with date display options which provide partial date functionality.
- **Text.**

# Numeric Formats

The four types of numeric formats are integer, floating-point double-precision, floating-point single-precision, and packed decimal. See *Numeric Display Options* on page 4-20 for additional information about numeric formats.

## Integer Format

You can use integer format for whole numbers—that is, any value composed of the digits zero to nine, without a decimal point.

You can also use integer fields with date display options to provide limited date support. This use of integer fields is described in the *Alphanumeric and Numeric Formats with Date Display Options* on page 4-33.

The integer FORMAT type is I. Display options are described in *Numeric Display Options* on page 4-20. The format of the length specification is

*n*

where:

*n*
Is the maximum number of digits, up to nine digits.

For example:

| Format | Display |
|---|---|
| I6 | 4316 |
| I2 | 22 |
| I4 | -617 |

## Floating-Point Double-Precision Format

You can use floating-point double-precision format for any number, including numbers with decimal positions—that is, for any value composed of the digits zero to nine and an optional decimal point.

The floating-point double-precision FORMAT type is D. The compatible display options are described in *Numeric Display Options* on page 4-20. The length specification format is

`t[.s]`

where:

*t*

Is the maximum number of characters to be displayed, up to a maximum of 16, including digits, a leading minus sign if the field will contain any negative values, and an optional decimal point if you want one to be displayed.

*s*

Is the number of digits that will follow the decimal point.

For example:

| Format | Display |
|---|---|
| D8.2 | 3,187.54 |
| D8 | 416 |

In the case of D8.2, the 8 represents the maximum number of places including the decimal point and decimal places. The 2 represents how many of these eight places are decimal places. The commas are automatically included in the display, and are not counted in the total.

## Floating-Point Single-Precision Format

You can use floating-point single-precision format for any number, including numbers with decimal positions—that is, for any value composed of the digits 0 to 9, including an optional decimal point. This format is intended for use with smaller decimal numbers: unlike floating-point double-precision format, its length cannot exceed nine positions.

The floating-point single-precision FORMAT type is F. The compatible display options are described in *Numeric Display Options* on page 4-20. The length specification format is

*t*[.*s*]

where:

*t*
: Is the maximum number of characters to be displayed, up to a maximum of 9, including digits, a leading minus sign if the field will contain any negative values, and an optional decimal point if you want one to be displayed.

*s*
: Is the number of digits that will follow the decimal point.

For example:

| Format | Display |
|---|---|
| F5.1 | 614.2 |
| F4 | 318 |

## Packed-Decimal Format

You can use packed-decimal format for any number, including decimal numbers—that is, for any value composed of the digits zero to nine, including an optional decimal point.

You can also use packed-decimal fields with date display options to provide limited date support. This use of packed-decimal fields is described in *Alphanumeric and Numeric Formats with Date Display Options* on page 4-33.

The packed-decimal FORMAT type is P. The compatible display options are described in *Numeric Display Options* on page 4-20.

The length specification format is

*m.n*

where:

*m*

Is the maximum number of characters to be displayed, up to a maximum of 33 positions (which include a position for the sign and decimal place).

*n*

Is the number of digits that will follow the decimal point. It can be up to 31 digits.

For example:

| Format | Display |
|---|---|
| P9.3 | 4168.368 |
| P7 | 617542 |

## Numeric Display Options

Display options may be used to edit numeric formats in various ways. Display options affect only how the data in the field is printed or appears on the screen. Display options do not affect how the data is stored in your file.

| Edit Option | Meaning | Effect |
|---|---|---|
| B | Bracket negative | Encloses negative numbers in parentheses. |
| C | Comma edit | Inserts a comma after every third significant digit, or a period instead of a comma if continental decimal notation is in use. |
| DMY | Day-Month-Year | Displays alphanumeric or integer data as a date in the form day/month/year. |
| E | Scientific notation | Displays only significant digits. |
| L | Leading zeroes | Adds leading zeroes. |
| M | Floating $ | Places a floating dollar sign $ to the left of the highest significant digit. |
| MDY | Month-Day-Year | Displays alphanumeric or integer data as a date in the form month/day/year. |
| N | Fixed $ | Places a dollar sign $ to the left of the field. |
| R | Credit (CR) negative | Places CR after negative numbers. |
| S | Zero suppress | If the data value is zero, prints a blank in its place. |
| T | Month translation | Displays the month as a three-character abbreviation. |
| YMD | Year-Month-Day | Displays alphanumeric or integer data as a date in the form year/month/day. |

The following table shows examples of the display options that are available for numeric fields.

| Option | Format | Data | Display |
|---|---|---|---|
| Comma inclusion | I6C | 41376 | 41,376 |
| Zero suppression | D6S | 0 | |
| Bracket negatives | I6B | -64187 | (64187) |
| Credit negative | I8R | -3167 | 3167 CR |
| Leading zeros | F4L | 31 | 0031 |
| Floating dollar | D7M | 6148 | $6,148 |
| Non-floating dollar | D7N | 5432 | $  5,432 |
| Scientific notation | D12.5E | 1234.5 | 0.123456D+04 |
| Year/month/day | I6YMD<br>I8YYMD | 980421<br>19980421 | 98/04/21<br>1998/04/21 |
| Month/day/year | I6MDY<br>I8MDYY | 042198<br>04211998 | 04/21/98<br>04/21/1998 |
| Day/month/year | I6DMY<br>I8DMYY | 210498<br>21041998 | 21/04/98<br>21/04/1998 |
| Month translation | I2MT | 07 | JUL |

Several display options can be combined, as shown:

| Format | Data | Display |
|---|---|---|
| I5CB | -61874 | (61,874) |

All of the options may be specified in any order. Options M and N (floating and non-floating dollar sign) and data format D (floating-point double-precision) automatically invoke option C (comma). Options L and S cannot be used together. Option T (translate month) can be included anywhere in an alphanumeric or integer USAGE specification that includes the M (month) display option. Date display options (D, M, T, and Y), which cannot be used with floating-point fields, are described in *Alphanumeric and Numeric Formats with Date Display Options* on page 4-33.

# Alphanumeric Format

You can use alphanumeric format for any value to be interpreted as a sequence of characters and composed of any combination of digits, letters, and other characters.

You can also use alphanumeric fields with date display options to provide limited date support. This use of alphanumeric fields is described in *Alphanumeric and Numeric Formats with Date Display Options* on page 4-33.

The alphanumeric FORMAT type is A. The format of the length specification is n, where n is the maximum number of characters in the field, up to 256 characters.

For example:

| Format | Display |
|---|---|
| A115 | The minutes of today's meeting were submitted... |
| A2 | B3 |
| A24 | 127-A429-BYQ-49 |

The standard numeric display options are not available for the alphanumeric data format. However, alphanumeric data can be printed under the control of a pattern that is supplied at run time. For instance, if a product code is to be displayed in parts, with each part separated by a "-", the following could be included at report time in a DEFINE FILE command

```
PRODCODE/A11 = EDIT (xxxxx,'999-999-999') ;
```

where:

*xxxxx*
Is the existing field name, not the newly defined field name.

If the value is 716431014, PRODCODE will be displayed as 716-431-014. See the *Creating Reports* manual for more information.

# Date Format

Date format enables you to define a field as a date and manipulate the field's value and display that value in ways appropriate to a date. Using date format, you can:

- Define date components such as year, quarter, month, day, and day of week, and extract them easily from date fields.
- Sort reports into date sequence, regardless of how the date is displayed.
- Do arithmetic with dates and compare dates without resorting to special date-handling functions.
- Refer to dates in a natural way, such as JAN 1 1995, without regard to display or editing formats.
- Automatically validate dates in transactions.

## Date Format Display Options

The date format does not specify type or length; instead, it specifies date component options (D, W, M, Q, Y, and YY) and display options. These options are shown in the following chart.

| Display Option | Meaning | Effect |
|---|---|---|
| `D` | Day | Prints a value from 1 to 31 for the day. |
| `M` | Month | Prints a value from 1 to 12 for the month. |
| `Y` | Year | Prints a two-digit year. |
| `YY` | Four-digit year | Prints a four-digit year. |
| `T` | Translate month or day | Prints a three-letter abbreviation for months in uppercase, if “M” is included in the FORMAT specification. |
| `t` | Translate month or day | Functions the same as uppercase T (described above), except that the first letter of the month or day is uppercase and the following letters are lowercase.* |
| `TR` | Translate month or day | Functions the same as uppercase T (described above), except that the entire month or day name is printed instead of an abbreviation. |
| `tr` | Translate month or day | Functions the same as lowercase t (described above), except that the entire month or day name is printed instead of an abbreviation.* |
| `Q` | Quarter | Prints the quarter (1 - 4 if Q is specified by itself, or Q1 - Q4 if it is specified together with other date format items such as Y). |
| `W` | Day-of-Week | If it is included in a FORMAT specification with other date component options, prints a three-letter abbreviation of the day of the week in uppercase. If it is the only date component option in the FORMAT specification, it prints the number of the day of the week (1-7, Mon=1). |
| `w` | Day-of-Week | Functions as uppercase W (described above), except that the first letter is uppercase and the following letters are lowercase.* |

| Display Option | Meaning | Effect |
|---|---|---|
| WR | Day-of-Week | Functions the same as uppercase W (described above), except that the entire day name is printed instead of an abbreviation.* |
| wr | Day-of-Week | Functions the same as lowercase w (described above), except that the entire day name is printed instead of an abbreviation.* |
| JUL | Julian format | Prints date in Julian format. |
| YYJUL | Julian format | Prints a Julian format date in the format YYYYDDD. The 7-digit format displays the four-digit year and the number of days counting from January 1. For example, January 3, 2001 in Julian format is 2001003. |

***Note:** When using these display options, be sure they are actually stored in the Master File as lowercase letters. To store characters in lowercase when using TED, you must first issue the command CASE M on the TED command line.

*Reference*

### How Are Field Formats of Y, YY, M, and W Stored?

The Y, YY, and M formats are not smart dates. Smart date formats YMD and YYMD, are stored as an offset from the base date of 12/31/1900. Smart date formats YM, YQ, YYM, and YYQ are stored as an offset from the base date 01/1901. W formats are stored as integers with a display length of one, containing values 1-7 representing the days of the week. Y, YY, and M formats are stored as integers. Y and M have display lengths of two. YY has a display length of four. When using Y and YY field formats, keep in mind these two important points:

- The Y formats will not sort based on DEFCENT and YRTHRESH settings. A field with a format of Y will not equal a YY field, as this is not a displacement, but a 4-digit integer.
- It is possible to use DEFCENT and YRTHRESH to convert a field from Y to YY format.

## Date Literals Interpretation Table

This table illustrates the behavior of FOCUS date formats. The columns indicate the number of input digits for a date format. The rows indicate the usage or format of the field. The intersection of row and column describes the result of input and format.

| **Date Format** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|
| YYMD | * | * | CC00/0m/dd | CC00/mm/dd |
| MDYY | * | * | * | * |
| DMYY | * | * | * | * |
| YMD | * | * | CC00/0m/dd | CC00/mm/dd |
| MDY | * | * | * | * |
| DMY | * | * | * | * |
| YYM | CC00/0m | CC00/mm | CC0y/mm | CCyy/mm |
| MYY | * | * | * | * |
| YM | CC00/0m | CC00/mm | CC0y/mm | CCyy/mm |
| MY | * | * | 0m/CCyy | mm/CCyy |
| M | 0m | mm | * | * |
| YYQ | CC00/q | CC0y/q | CCyy/q | 0yyy/q |
| QYY | * | * | q/CCyy | * |
| YQ | CC00/q | CC0y/q | CCyy/q | 0yyy/q |
| QY | * | * | q/CCyy | * |
| Q | q | * | * | * |
| JUL | CC00/00d | CC00/0dd | CC00/ddd | CC0y/ddd |
| YY | 000y | 00yy | 0yyy | yyyy |
| Y | 0y | yy | * | * |
| D | 0d | dd | * | * |
| W | w | * | * | * |

| Date Format | 5 | 6 | 7 | 8 |
|---|---|---|---|---|
| YYMD | CC0y/mm/dd | CCyy/mm/dd | 0yyy/mm/dd | yyyy/mm/dd |
| MDYY | 0m/dd/CCyy | mm/dd/CCyy | 0m/dd/yyyy | mm/dd/yyyy |
| DMYY | 0d/mm/CCyy | dd/mm/CCyy | 0d/mm/yyyy | dd/mm/yyyy |
| YMD | CC0y/mm/dd | CCyy/mm/dd | 0yyy/mm/dd | yyyy/mm/dd |
| MDY | 0m/dd/CCyy | mm/dd/CCyy | 0m/dd/yyyy | mm/dd/yyyy |
| DMY | 0d/mm/CCyy | dd/mm/CCyy | 0d/mm/yyyy | dd/mm/yyyy |
| YYM | 0yyy/mm | yyyy/mm | * | * |
| MYY | 0m/yyyy | mm/yyyy | * | * |
| YM | 0yyy/mm | yyyy/mm | * | * |
| MY | 0m/yyyy | mm/yyyy | * | * |
| M | * | * | * | * |
| YYQ | yyyy/q | * | * | * |
| QYY | q/yyyy | * | * | * |
| YQ | yyyy/q | * | * | * |
| QY | q/yyyy | * | * | * |
| Q | * | * | * | * |
| JUL | CCyy/ddd | * | * | * |
| YY | * | * | * | * |
| Y | * | * | * | * |
| D | * | * | * | * |
| W | * | * | * | * |

- CC stands for two century digits provided by DFC/YRT settings.
- * stands for error message FOC177 (invalid date constant).
- FOCUS reads date literals from right to left.

## Controlling the Date Separator

You can control the date separators when the date is displayed. In basic date format, such as YMD and MDYY, the date components are displayed separated by a slash character (/). The same is true for the year-month format. Year-quarter format is displayed with the year and quarter separated by a blank (for example, 94 Q3 or Q3 1994). The single component formats display just the single number or name.

The separating character can be changed to a period, a dash, or a blank, or can even be eliminated entirely. The following table shows the FORMAT specifications that can be used to change the separating character.

| **Format** | **Display** |
|---|---|
| YMD | 93/12/24 |
| Y.M.D | 93.12.24 |
| Y-M | 93-12 |
| YBMBD | 93 12 24 (The letter “B” signifies blank spaces.) |
| Y\|M\|D | 931224 (The concatenation symbol “\|” eliminates the separation character.) |

## Date Translation

Numeric months and days can be replaced by a translation, such as JAN, January, Wed, or Wednesday. The translated month or day can be abbreviated to three characters or fully spelled out. It can appear in either uppercase or lowercase. In addition, the day of the week (for example, Monday) can be appended to the beginning or end of the date. All of these options are independent of each other.

| **Translation** | **Display** |
|---|---|
| `MT` | `JAN` |
| `Mt` | `Jan` |
| `MTR` | `JANUARY` |
| `Mtr` | `January` |
| `WR` | `MONDAY` |
| `wr` | `Monday` |

## Sample Date Formats

The following chart shows some sample USAGE and ACTUAL formats for data stored in a non-FOCUS data source. The Value column shows the actual data value and the Display column shows how the data is displayed.

| USAGE | ACTUAL | Value | Display |
|---|---|---|---|
| wrMtrDYY | A6YMD | 990315 | Monday, March 15, 1999 |
| YQ | A6YMD | 990315 | 99 Q1 |
| QYY | A6YMD | 990315 | Q1 1999 |
| YMD | A6 | 990315 | 99/03/15 |
| MDYY | A6YMD | 990315 | 03/15/1999 |

Note that the date attributes in the ACTUAL format specify the order in which the date is stored in the external file. If the ACTUAL format does not specify the order of the month, day, and year, it will be inferred from the USAGE format, as illustrated in the fourth example.

## Using Date Fields

A field formatted as a date is automatically validated when it is entered. It can be entered as a natural date literal (for example, JAN 12 1999) or as a numeric date literal (for example, 011299).

Natural date literals, by including spaces between date components and using abbreviations of month names, enable you to specify a date in a natural, easily understandable way. For example, April 25, 1999 can be specified as any of the following natural date literals:

```
APR 25 1999
25 APR 1999
1999 APR 25
```

Natural date literals can be used in all date computations and with the PROMPT, FREEFORM, and FIXFORM statements when updating a file. Examples are shown in the following chart.

```
In WHERE screening                    WHERE MYDATE IS 'APR 25 1999'
In arithmetic expressions             MYDATE - '1999 APR 25'
In computational date comparisons     IF MYDATE GT '25 APR 1999'
In replies to MODIFY prompts          MYDATE==> APR 25 1999
In comma-delimited data               ...,MYDATE = APR 25 1999, ...
```

Note that natural date literals cannot be used to enter dates using FIDEL.

The following chart describes the format of natural date literals.

| Literal | Format |
|---|---|
| Year-month-day | Four-digit year; uppercase three-character abbreviation, or uppercase full name, of the month; and one- or two-digit day of the month (for example, 1999 APR 25 or APRIL 25 1999). |
| Year-month | Year and month as described above. |
| Year-quarter | Year as described above, Q plus quarter number for quarter (for example, 1999 Q3). |
| Month | Month as described above. |
| Quarter | Quarter as described above. |
| Day of week | Three-character, uppercase abbreviation, or full, uppercase name, of the day (for example, MON or MONDAY). |

The date components of a natural date literal can be specified in any order, regardless of their order in the FORMAT specification of the target field. Date components are separated by one or more blanks.

For example, if a FORMAT specification for a date field is YM, a natural date literal written to that field can include the year and month in any order. MAY 1999 and 1990 APR would both be valid literals.

## Numeric Date Literals

Numeric date literals differ from natural date literals in that they are simple strings of digits. The order of the date components in a numeric date literal must match the order of the date components in the corresponding FORMAT specification. In addition, the numeric date literal must include all of the date components included in the FORMAT specification. For example, if the FORMAT specification is DMY, then April 25 1999 must be represented as:

```
250499
```

Numeric date literals can be used in all date computations and all methods of file updating.

## Date Fields in Arithmetic Expressions

The general rule for manipulating date fields in arithmetic expressions is that date fields in the same expression must specify the same date components. The date components can be specified in any order, and display options are ignored. Valid date components are Y or YY, Q, M, W, and D.

Note that arithmetic expressions assigned to quarters, months, or days of the week are computed modulo 4, 12, and 7, respectively, so that anomalies like fifth quarters and thirteenth months are avoided.

For example, if NEWQUARTER and THISQUARTER both have FORMAT specifications of Q, and the value of THISQUARTER is 2, then the following statement

```
NEWQUARTER = THISQUARTER + 3
```

gives NEWQUARTER a value of 1 (that is, the remainder of 5 divided by 4).

## Converting Date Fields

You can also convert date fields. Two types of conversion are possible: format conversion and date component conversion. In the first case, the value of a date format field can be assigned to an alphanumeric or integer field that uses date display options (see the following section); the reverse conversion is also possible.

In the second case, a field whose FORMAT specifies one set of date components can be assigned to another field specifying different date components.

For example, the value of REPORTDATE (DMY) can be assigned to ORDERDATE (Y); in this case, the year is being extracted from REPORTDATE. If REPORTDATE is Apr 27 89, ORDERDATE is 89.

You can also assign the value of ORDERDATE to REPORTDATE; if the value of ORDERDATE is 89, the value of REPORTDATE would be Jan 1 89. In this case, REPORTDATE is given values for the missing date components.

*Syntax*

### How to Convert Date Fields

The general formula for converting date fields is:

```
field1/format = field2;
```

where:

*field1*
Is a date format field, or an alphanumeric or integer format field using date display options.

*format*
Is the FORMAT (or USAGE) specification of *field1* (the target field).

*field2*
Is a date format field, or an alphanumeric or integer format field using date display options. The format types (alpha, integer, or date) and the date components (YY, Y, Q, M, W, D) of *field1* and *field2* do not need to match.

### Displaying Non-Standard Date Formats

In previous releases of FOCUS, if a date field in a non-FOCUS file contained an invalid date, a diagnostic error was displayed and the entire record failed to display in a report. For example, if a date field contained ‘980450’ with an ACTUAL of A6 and a USAGE of YMD, the record containing that field would not display. With the use of a new command, it is possible to display the rest of the record that contains the incorrect date.

*Syntax*

### Invoking ALLOWCVTERR

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

where:

ON
Allows the display of a field containing an incorrect date.

OFF
Generates an error if incorrect data is encountered, and does not display the record containing the bad data. This is the default value.

When it encounters a bad date, ALLOWCVTERR sets the value of the field to either MISSING or to the base date. When a field is being converted and ALLOWCVTERR is set on, FOCUS first checks to see if MISSING=ON. The following chart shows the results of interaction between DATEDISPLAY and MISSING assuming ALLOWCVTERR=ON and the presence of a bad date.

| | MISSING=OFF | MISSING=ON |
|---|---|---|
| DATEDISPLAY=ON | Displays Base Date 19001231 or 1901/1 | . |
| DATEDISPLAY=OFF | Displays Blanks | . |

DATEDISPLAY only affects how the base date is displayed. See the *Developing Applications* manual for a description of DATEDISPLAY.

## How Date Fields Are Represented Internally

FOCUS represents date fields internally as four-byte binary integers indicating the elapsed time since the date format base date. For each field, the unit of elapsed time is that field's smallest date component.

For example, if the FORMAT specification of REPORTDATE is MDY, then elapsed time is measured in days, and internally the field contains the number of days elapsed between the entered date and the base date. If you entered the numeric literal for February 13, 1964 (that is, 021364), and then printed the field in a report, 02/13/64 would be displayed. If you used it in the equation

```
NEWDATE = 'FEB 28 1964' - REPORTDATE ;
DAYS/D = NEWDATE ;
```

then the value of DAYS would be 15. However, the internal representation of REPORTDATE would be a four-byte binary integer representing the number of days between December 31, 1900 and February 13, 1964.

Just as the unit of elapsed time is based on a field's smallest date component, so too is the base date. For example, for a YQ field, elapsed time is measured in quarters and the base date is the first quarter of 1901. For a YM field, elapsed time is measured in months and the base date is the first month of 1901.

In reports, to display blanks or the actual base date, use the SET DATEDISPLAY command described in the *Developing Applications* manual. The default value, OFF, displays blanks when a date matches the base date. ON displays the actual base date value.

You do not need to be concerned with the date format's internal representation, except to note that all dates set to the base date display as blanks, and all date fields that are entered blank or as all zeros are accepted during validation and interpreted as the base date. They will be displayed as blanks, but will be interpreted in date computations and expressions as the base date.

## Date Format Support

Date format fields are used in special ways with the following FOCUS facilities:

- **Dialogue Manager.** Amper variables can function as date fields if they are set to natural date literals. For example:

  ```
  -SET &NOW = 'APR 25 1960' ;
  -SET &LATER = '1990 25 APR' ;
  -SET &DELAY = &LATER - &NOW ;
  ```

  In this case, the value of &DELAY is the difference between the two dates, measured in days: 10,957.

- **Extract files.** Date fields in SAVB and unformatted HOLD files are stored as four-byte binary integers representing the time difference between the field's face value and the standard FOCUS base date. Date fields in SAVE files and formatted HOLD files (for example, FORMAT WP) are stored in internal format, without any display options.

- **GRAPH.** Date fields are not supported as sort fields in ACROSS and BY phrases.

- **EMR.** Date fields are not supported within the RECAP statement.

## Alphanumeric and Numeric Formats With Date Display Options

In addition to the standard date format, you can also represent a date by using an alphanumeric, integer, or packed-decimal field with date display options (D, M, Y, and T). Note, however, that this does not offer the full date support that is provided by the standard date format.

Alphanumeric and integer fields used with date display options have some date functionality when used with special date functions, as described in the *Creating Reports* manual.

When representing dates as alphanumeric or integer fields with date display options, you can specify the year, month, and day. If all three of these elements are present, then the date has six digits (or eight if the year is presented as four digits) and the FORMAT can be:

| Format | Display |
|---|---|
| I6MDY | 04/21/98 |
| I6YMD | 98/04/21 |
| P6DMY | 21/04/98 |
| I8DMYY | 21/04/1998 |

A month's number (1 to 12) can be translated to the corresponding month name by adding the letter "T" to the format, immediately after the "M." For instance:

| Format | Data | Display |
|---|---|---|
| I6MTDY | 05/21/98 | MAY 21 98 |
| I4MTY | 0698 | JUN 98 |
| I2MT | 07 | JUL |

If the date has only the month element, a format of I2MT will display the value 4 as APR, for example. This is particularly useful in reports where columns or rows are sorted by month. They will then appear in correct calendar order; for example, JAN, FEB, MAR, because the sorting is based on the numerical, not alphabetical, values. (Note that without the T display option, I2M would be interpreted as an integer with a floating dollar sign.)

# Text Field Format

Text fields are assigned a format of TXnn[F]. The syntax for defining a text field in a Master File is

```
FIELD = fieldname, ALIAS = aliasname, FORMAT = TXnn,$
```

or

```
FIELD = fieldname, ALIAS = aliasname, FORMAT = TXnnF,$
```

where:

*fieldname*
: Is the name you assign the text field.

*aliasname*
: Is an alternate name for the field name.

*nn*
: Is the output display length in TABLE for the text field. The display length may be between 1 and 256 characters.

F
: Is used to format the text field for redisplay when TED is called using ON MATCH or ON NOMATCH. When F is specified, the text field is formatted as TX80 and is displayed. When F is not specified, the field is redisplayed exactly as entered.

For example, the text field in the COURSES file is specified as:

```
FIELD = DESCRIPTION, ALIAS = CDESC, USAGE = TX50,$
```

All letters, digits and special characters can be stored with this format. The following are some sample text field formats.

| Format | Display |
|---|---|
| TX50 | This course provides the DP professional with the skills needed to create, maintain, and report from FOCUS databases. |
| TX35 | This course provides the DP professional with the skills needed to create, maintain, and report from FOCUS databases. |

The standard edit options are not available for the text field format.

# Sliding Window Field Attributes for Handling Two-Digit Years

If your data source contains date values stored with two-digit years, you can use the DEFCENT and YRTHRESH attributes to assign century values to these fields. See the *Developing Applications* manual for a description of the sliding window approach and for related SET command parameters. You can also set sliding window attributes at the file level as described in *Sliding Window File Attributes for Handling Two-Digit Years* in Chapter 2, *Describing Data Sources*.

## *Syntax* How to Establish Cross-Century Dates for a Field in a Master File

At the field level, the settings DEFCENT and YRTHRESH were added. The DEFCENT syntax is

```
{DEFCENT|DFC} = cc
```

where:

*cc*
Is a two-digit century value. The default value is 19.

The YRTHRESH syntax is

```
{YRTHRESH|YRT} = {yy|-nn}
```

where:

*yy*

Is the year for which the century value changes to DEFCENT. Below this value, the century is DEFCENT+1. The default value is zero.

*-nn*

Sets YRTHRESH to an offset from the current year. The default value is zero.

You decide the number of years to offset in YRTHRESH. For example, if the current year is 1998 and you wish to set YRTHRESH to 60, you can set YRTHRESH to -38 (1998 - 38 = 60). By setting YRTHRESH to a negative number FOCUS subtracts that value from the current year. In the year 1999, YRTHRESH is 61 instead of 60 (1999 - 38 = 61) illustrating how the moving window functions without outside intervention.

If you set YRTHRESH to a value that crosses a century boundary, the value of DEFCENT is recalculated. For example, if you set YRTHRESH to minus 99 (1998-99 = -1), DEFCENT is calculated to 18 and YRTHRESH becomes 99. The 100-year span begins with a pivot year of 1899 and ends with year 1998. ? SET and ? SET ALL reflect the new DEFCENT setting. **Note:** When you use this option, do not specify DEFCENT. It is calculated automatically.

### *Example* Establishing Cross-Century Dates at the Field Level

```
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, DEFCENT=19, YRTHRESH=75,$
```

To see the application of DEFCENT and YRTHRESH to interpret two-digit years, consider the following:

```
DEFCENT=19, YRTHRESH=80
```

describes a range from 1980 to 2079. If a two-digit year field contains 99, the year is 1999. If it is 79, the year is 2079. If it is 00, the year is 2000.

*Reference* **Special Considerations**

- Field level settings take precedence over file level settings.
- Compiled MODIFYs from releases earlier than Version 7.0 Release 6 must be recompiled to use this feature.
- ON TABLE SET is not supported.
- In HOLD Master Files, the concatenation of FILE and FIELD level information is propagated as FIELD level.
- With DEFINE and COMPUTE, both DFC and YRT must be specified. Each can be specified separately in the Master File.

# Null or MISSING Values: MISSING

If a segment instance exists but no data has been entered into one of its fields, that field has no value. Some types of data sources represent this absence of data as a blank space ( ) or zero (0), but others explicitly indicate an absence of data with a null indicator or as a special null value. Null values (sometimes known as missing data) are significant in reporting applications, especially those that perform aggregating functions such as averaging.

If your type of data source supports missing data, as do FOCUS databases and most relational data sources, then you can use the optional MISSING attribute to enable null values to be entered into and read from a field. MISSING plays a role when you use FOCUS to:

- **Create new segment instances.** If no value is supplied for a field for which MISSING has been set to ON, then FOCUS assigns the field a missing value.
- **Generate reports.** If FOCUS reads a field with a null value, it ignores that value when it performs aggregating calculations such as averaging and summing. If the report calls for the field's value to be displayed, FOCUS displays a special character that indicates a missing value. The default character is a period (.), but you can change it to any character string you wish using the SET NODATA command, as described in the *Developing Applications* manual.

*Syntax* **How to Specify Missing Values**

The syntax of the MISSING attribute is

```
MISSING = {ON|OFF}
```

Valid values are:

`ON`

Enables FOCUS to distinguish a missing value from an intentionally entered blank or zero when creating new segment instances and reporting.

`OFF`

Tells FOCUS not to distinguish between missing values and blank or zero values when creating new segment instances and reporting. This is the default value.

### *Reference* Usage Notes for MISSING

Note the following rules when using MISSING:

- **Alias.** MISSING does not have an alias.
- **Setting.** It is recommended that you set the MISSING attribute to match the field's predefined null characteristic (whether the characteristic was explicitly set when the data source was created, or set by default). For example, if a relational table column has been created with the ability to accept null data, you should describe the field to FOCUS with the MISSING attribute set to ON so that FOCUS correctly interprets its null values.

  This is not a consideration for FOCUS databases, for which the field declaration in the Master File both defines the field and describes it to FOCUS.
- **Changes.** You can change the MISSING attribute at any time. Note that changing MISSING will not affect the actual stored data values that had been entered using FOCUS under the old setting. However, it will affect how that data is interpreted by FOCUS: if null data is entered when MISSING is set to ON, and then MISSING is switched to OFF, the data originally entered as null will be interpreted as blanks (for alphanumeric fields) or zeros (for numeric fields). The only exception is FOCUS databases, in which the data originally entered as missing will be interpreted as the FOCUS internal missing value for that data type, which is described in Chapter 7, *Describing FOCUS Databases*.

## Using Missing Values

Consider the field values shown in the following four records:

| | | | |
|---|---|---|---|
| | | 1 | 3 |

If you average these values without declaring the field with the MISSING attribute, FOCUS will automatically supply a value of zero for the two blank records. Thus, the average of these four records will be (0+0+1+3)/4 or 1. If you set MISSING to ON, FOCUS will ignore the two blank records, so your average will be (1+3)/2 or 2.

Missing values in a unique segment are also automatically supplied with a zero or a blank. What distinguishes missing values in unique segments from others is that they are not stored. You do have to supply a missing attribute for fields in unique segments on which you want to perform counts or averages.

The *Creating Reports* manual contains a more thorough discussion of using null values (sometimes called missing data) in reports. Included in the discussion are alternative ways of distinguishing these values in reports, such as using the WHERE phrase with MISSING selection operators, and creating temporary fields using the DEFINE FILE command with the SOME or ALL phrase.

# Describing Temporary Fields: DEFINE

DEFINE is an optional attribute used to create temporary fields for reporting. You can derive the temporary field's value from information already in the data source—that is, from permanent fields. Some common uses of temporary data fields include:

- Computing new numerical values that are not on the data record.
- Computing a new string of alphanumeric characters from other strings.
- Classifying data values into ranges or groups.
- Invoking subroutines in calculations.

Temporary fields are available whenever the data source is used for reporting.

## *Syntax* How to Define a Temporary Field

The syntax is

```
DEFINE fieldname/format = expression; [, attribute2, ... ] $
```

where:

*fieldname*

Is the name of the temporary field. You can assign any name up to 66 characters long. The name is subject to the same conventions as names assigned using the FIELDNAME attribute. FIELDNAME is described in *The Field's Name: FIELDNAME* on page 4-2.

*format*

Is the field's format. The format is specified in the same way as formats assigned using the FORMAT attribute, which is described in *The Displayed Data Type: FORMAT* on page 4-14. If you do not specify a format, it defaults to D12.2.

*expression*

Is a valid FOCUS expression. FOCUS expressions are fully described in the *Creating Reports* manual. The expression must end with a semicolon (;).

Note that when an IF-THEN phrase is used in the expression of a temporary field, it must include the ELSE phrase.

*attribute2*

The declaration for a temporary field can include additional optional attributes, such as TITLE and DESCRIPTION. Any additional attributes must be on the same line as the semicolon that ends the DEFINE expression. An exception is made for the DESCRIPTION attribute; however, if you put it on the final line of the declaration, it does not need to be on the same line as the semicolon.

You can devote an entire line to these additional attributes by placing the semicolon on the line following the DEFINE expression.

Place each DEFINE attribute after all of the field descriptions for that segment. For example, the following shows how to define a field called PROFIT in the segment CARS:

```
SEGMENT = CARS ,SEGTYPE = S1 ,PARENT = CARREC, $
    FIELDNAME = DEALER_COST ,ALIAS = DCOST ,USAGE = D7, $
    FIELDNAME = RETAIL_COST ,ALIAS = RCOST ,USAGE = D7, $
    DEFINE PROFIT/D7 = RETAIL_COST - DEALER_COST; $
```

### *Reference* Usage Notes for DEFINE

Note the following rules when using DEFINE:

- **Alias.** DEFINE does not have an alias.
- **Changes.** You can change the temporary field's declaration at any time.

## Using Temporary Fields

A DEFINE statement cannot contain qualified field names on the left-hand side of the expression. You can use the WITH phrase on the left-hand side to place the defined field in the same segment as any real field you choose.

When FIELDNAME is set to OLD, a DEFINE attribute in a Master File can only refer to fields in its own segment. In this case, if you want to create a DEFINE field that uses information from several different segments, you will have to create it with a DEFINE FILE request prior to a report request as discussed in the *Creating Reports* manual.

When FIELDNAME is set to NEW, expressions on the right-hand side of the DEFINE can refer to fields from any segment in the same path. The expression on the right-hand side of a DEFINE or REDEFINES statement in a Master File can contain qualified field names.

A DEFINE attribute in a Master File can refer to only fields in its own path. If you want to create a temporary field that derives its value from fields in several different segments, you will have to create it with a DEFINE FILE command prior to a report request, as discussed in the *Creating Reports* manual. The DEFINE FILE command is also helpful when you wish to create a temporary field that will be used only once, and you do not want to add a declaration for it to the Master File.

Temporary fields defined in the Master File are available whenever the data source is used and are treated like other stored fields. Thus, a field defined in the Master File cannot be cleared in your report request. A temporary field cannot be used for cross-referencing in a join.

**Note:** Maintain does not support DEFINE attributes that have a constant value. Using such a field in a Maintain procedure generates the following error message:

```
(FOC03605) name is not recognized.
```

# Validating Data: ACCEPT

ACCEPT is an optional attribute that you can use to validate data as it is entered into a field from a MODIFY or FSCAN procedure. The ACCEPT test is applied immediately after a CRTFORM, PROMPT, FIXFORM, or FREEFORM is processed after which subsequent COMPUTE statements can manipulate the value. By including ACCEPT in a field declaration you can define a list or range of acceptable field values. In relational terms, you are defining the domain.

**Note:** Suffix VSAM and FIX files may use the ACCEPT attribute to specify multiple RECTYPE values, which are discussed in Chapter 5, *Describing Sequential Data Files*.

## *Syntax* How to Validate Data

The syntax for the ACCEPT attribute is

```
ACCEPT = list
ACCEPT = range
ACCEPT = FIND (field [AS name] IN file)
```

where:

*list*

Is a string of acceptable values. The syntax is:

```
value1 OR value2 OR value3...
```

For example, ACCEPT = RED OR WHITE OR BLUE. You can also use a blank as an item separator. If the list of acceptable values runs longer than one line, continue it on the next. The list is terminated by a comma.

*range*

Gives the range of acceptable values. The syntax is:

```
value1 TO value2
```

For example, ACCEPT = 150 TO 1000.

FIND

Verifies the incoming data against the values in another indexed field. This option is available only for FOCUS databases; see Chapter 7, *Describing FOCUS Databases*, for more information.

Any value in the ACCEPT that contains an embedded blank (for example, Great Britain) must be enclosed within single quotation marks. For example:

```
ACCEPT = SPAIN OR ITALY OR FRANCE OR 'GREAT BRITAIN'
```

If the ACCEPT attribute is included in a field declaration and the SET command parameter ACCBLN has a value of OFF, blank ( ) and zero (0) values will be accepted only if they are explicitly coded into the ACCEPT. SET ACCBLN is described in the *Developing Applications* manual.

### *Reference* Usage Notes for ACCEPT

Note the following rule when using ACCEPT:

- **Alias.** ACCEPT does not have an alias.
- **Changes.** You can change the information in an ACCEPT attribute at any time.
- **Temporary fields.** You cannot use the ACCEPT attribute to validate temporary fields created with the DEFINE attribute.
- **HOLD files.** If you wish the ACCEPT attribute to be propagated into the Master File of a HOLD file, use the SET HOLDATTR command. HOLD files are discussed in the *Creating Reports* manual.
- **ACCEPT** is used only in MODIFY procedures. It is useful for providing one central validation list to be used by several procedures. The FIND function is useful when the list of values is large or undergoes frequent change.
- **The HELPMESSAGE attribute** defines a message to display based on the results of an ACCEPT test.

## Online Help Information: HELPMESSAGE

HELPMESSAGE is an optional field attribute. It enables you to include a one-line text message in the Master File. This text, or message, is displayed on one line in the TYPE area of MODIFY CRTFORMs. For example, you can include a message that lists valid values for a field, or one that provides information about the format of a field. The specified message is displayed when:

- The value entered for a data source field is invalid according to the ACCEPT test for the field.
- The value entered for a data source field causes a format error.
- The user places the cursor in the data entry area for a particular field and presses a predefined PF key.

The syntax for the HELPMESSAGE attribute in the Master File is

```
FIELDNAME = name, ALIAS = alias, FORMAT = format,
    HELPMESSAGE = text...,$
```

where:

*text*

Is one line of text, up to 78 characters long. All characters and digits are acceptable. Text containing a comma must be enclosed within single quotation marks. Leading blanks are ignored.

For example:

```
FIELDNAME = DEPARTMENT, ALIAS = DPT, FORMAT = A10,
    ACCEPT = MIS PRODUCTION SALES,
    HELPMESSAGE = 'DEPARTMENT MUST BE MIS, PRODUCTION, OR SALES',$
```

The ACCEPT attribute for the DEPARTMENT field causes values entered for that field to be tested. If the incoming value is not MIS, PRODUCTION, or SALES, a FOCUS error message is displayed. Then the specified HELPMESSAGE text is displayed:

```
(FOC534) THE DATA VALUE IS NOT AMONG THE ACCEPTABLE VALUES FOR DEPARTMENT
DEPARTMENT MUST BE MIS, PRODUCTION, OR SALES
```

**Note:** FOCUS error messages are displayed whether or not the HELPMESSAGE attribute is used. The HELPMESSAGE attribute also causes a message to be displayed when a format error occurs. For example, if the field HIRE_DATE is specified in the Master File as follows

```
FIELDNAME = HIRE_DATE, ALIAS = HDT, FORMAT = YMD,
    HELPMESSAGE = THE FORMAT FOR HIRE_DATE IS YMD,$
```

and alphabetic characters are entered for this field on a CRTFORM, the following message will appear on the screen:

```
FORMAT ERROR IN VALUE ENTERED FOR FIELD HIRE_DATE
    THE FORMAT FOR HIRE_DATE IS YMD
```

Note that the same message provided with the HELPMESSAGE attribute is displayed when either a format error or a failed ACCEPT test occurs.

### Setting a HELP (PF) Key

To see the HELPMESSAGE text for any field on the CRTFORM, use the SET command to define a PF key for HELP before executing the MODIFY program. Use the following (which is the alias for HELPMESSAGE) syntax

```
SET PFnn = HELP
```

where:

*nn*
Is the number of the PF key you wish to define.

To see a message for a field, position the cursor on the data entry area of that field and press the PF key defined for HELP. If no message has been defined for the field, you will see the following message:

```
NO HELP AVAILABLE FOR THIS FIELD
```

For a FOCUS database, the HELPMESSAGE attribute can be changed without rebuilding the database.

## Alternative Report Column Titles: TITLE

When you generate a report, each column title in the report defaults to the name of the field displayed in that column. However, you can change the default column title by specifying the optional TITLE attribute for that field.

Of course, you can always specify a different column title within an individual report by using the AS phrase in that report request, as described in the *Creating Reports* manual.

Note that the TITLE attribute has no effect in a report if the field is used with a prefix operator such as AVE. You can supply an alternative column title for fields used with prefix operators by using the AS phrase.

*Syntax*

## How to Specify Alternative Titles

The syntax for the TITLE attribute is

```
TITLE = 'text'
```

where:

*text*
: Is any string of up to 64 characters. You can split the text across as many as five separate title lines by separating the lines with a comma (,). You can include blanks at the end of a column title by including a slash (/) in the final blank position. You must enclose the string within single quotation marks.

For example

```
FIELD = LNAME, ALIAS = LN, FORMAT = A15, TITLE = 'Client,Name',$
```

replaces the default column heading, LNAME, with the following:

```
Client
Name
------
```

*Reference*

## Usage Notes for TITLE

Note the following rules when using TITLE:

- **Alias.** TITLE does not have an alias.
- **Changes.** You can change the information in TITLE at any time. You can also override the TITLE with an AS name in a request or the SET TITLE command.
- **Temporary fields.** If you use the TITLE attribute for a temporary field created with the DEFINE attribute, the semicolon (;) terminating the DEFINE expression must be on the same line as the TITLE keyword.
- **HOLD files.** If you wish the TITLE attribute to be propagated into the Master File of a HOLD file, use the SET HOLDATTR command. HOLD files are discussed in the *Creating Reports* manual.

# Documenting the Field: DESCRIPTION

DESCRIPTION is an optional attribute that enables you to provide comments and other documentation for a field within the Master File. You can include any comment up to 44 characters in length.

Note that you can also add documentation to a field declaration, or to a segment or file declaration, by typing a comment in the columns following the terminating dollar sign. You can even create an entire comment line by inserting a new line following a declaration and placing a dollar sign at the beginning of the line. The syntax and rules for creating a FOCUS Master File are described in *What Are FOCUS File Descriptions?* in Chapter 1, *Describing Data With FOCUS*.

The DESCRIPTION attribute for a FOCUS database can be changed at any time without rebuilding the file.

## *Syntax* How to Supply Field Documentation

The syntax for the DESCRIPTION attribute is

```
DESC[RIPTION] = text
```

where:

`DESCRIPTION`
: Can be shortened to DESC. Abbreviating the keyword has no effect on its function.

*`text`*
: Is any string of up to 44 characters. If the string contains a comma, the string must be enclosed within single quotation marks.

For example:

```
FIELD=UNITS,ALIAS=QTY,FORMAT=I6, DESC='QUANTITY SOLD, NOT RETURNED',$
```

## *Reference* Usage Notes for DESCRIPTION

Note the following rules when using the DESCRIPTION attribute:

- **Alias.** The DESCRIPTION attribute has an alias of DEFINITION.
- **Changes.** You can change DESCRIPTION at any time.
- **Temporary fields.** If you use the DESCRIPTION attribute for a temporary field created with the DEFINE attribute, the DESCRIPTION attribute must be on the same line as the semicolon (;) terminating the DEFINE expression or, if there are other attributes in the declaration (such as TITLE), on the last line of the declaration.

# 5 Describing Sequential Data Files

**Topics:**

- Sequential Data File Formats
- Standard Master File Attributes
- Describing Multiply-Occurring Fields in a Free-Format File
- Describing Multiply-Occurring Fields in a Fixed-Format File
- Describing Multiple Record Types
- Combining Multiply-Occurring Fields and Multiple Record Types

You can describe and report from sequential data files using FOCUS. In a sequential data file, records are stored and retrieved in the same order as they were entered.

## Sequential Data File Formats

FOCUS recognizes sequential data files formatted in two ways:

- Fixed-format, in which each field occupies a pre-defined position in the record.
- Free-format, also known as comma-delimited, in which fields can occupy any position in a record.

You can describe two types of sequential data files to FOCUS:

- **Simple.** This is the most basic type, consisting of only one segment. It is supported in both fixed- and free-format.
- **Complex.** This is a multi-segment data file. The descendant segments exist in the file as multiply-occurring fields (which are supported in both fixed- and free-format) or multiple record types (which are supported only in fixed-format).

# What Are Fixed-Format Data Files?

Fixed-format data files are sequential data files in which each field occupies a pre-defined position in the record. You describe the record format in the Master File.

For example, a fixed-format record might look like this:

```
1352334556George Eliot The Mill on the Floss H
```

The simplest form of a fixed-record file can be described by providing just field declarations (if you do not intend to use it in a JOIN request). For example, suppose you have a file for a library that consists of the following components:

- A number, like an ISBN number, that identifies the book by publisher, author, and title.
- The name of the author.
- The title of the book.
- A single letter that indicates whether the book is hard- or soft-bound.
- The book's price.
- A serial number that actually identifies the individual copies of the book in the library (a call number).
- A synopsis of the book.

This file can be described in FOCUS with just the seven field declarations shown here:

```
FIELDNAME = PUBNO   ,ALIAS = PN  ,USAGE = A10   ,ACTUAL = A10   ,$
FIELDNAME = AUTHOR  ,ALIAS = AT  ,USAGE = A25   ,ACTUAL = A25   ,$
FIELDNAME = TITLE   ,ALIAS = TL  ,USAGE = A50   ,ACTUAL = A50   ,$
FIELDNAME = BINDING ,ALIAS = BI  ,USAGE = A1    ,ACTUAL = A1    ,$
FIELDNAME = PRICE   ,ALIAS = PR  ,USAGE = D8.2N ,ACTUAL = D8    ,$
FIELDNAME = SERIAL  ,ALIAS = SN  ,USAGE = A15   ,ACTUAL = A15   ,$
FIELDNAME = SYNOPSIS,ALIAS = SYN ,USAGE = A150  ,ACTUAL = A150  ,$
```

**Note:**

- Each declaration begins with the word FIELDNAME, and normally contains four elements (a FIELDNAME, an ALIAS, a USAGE attribute, and an ACTUAL attribute).
- If you omit the optional ALIAS, its absence must be signaled by a second comma between FIELDNAME and USAGE (FIELDNAME=PUBNO,,A10,A10,$).
- Both the USAGE and the ACTUAL attributes must be included. Failure to specify both is a common cause of errors in describing external files to FOCUS (FOCUS data files do not have ACTUAL attributes).
- Each declaration must be terminated with a comma followed by a dollar sign (,$).
- When using Maintain to read a fixed-format data file, the record length as described in the Master File may not exceed the actual length of the data record (the LRECL value).

You need only describe to FOCUS those fields that you intend to refer to. This is very significant when using existing files, because they frequently contain information that you do not need for your FOCUS reports. You describe only the fields you wish to include in your reports or calculations and use filler fields to represent the rest of the logical record length (LRECL) of the file.

In the above example, the book synopsis is hardly necessary for most reports. The synopsis can, therefore, be replaced with a filler field, as follows:

```
FIELDNAME = FILLER, ALIAS = FILL1, USAGE = A150, ACTUAL = A150,$
```

Fillers of this form may contain up to 256 characters. If you need to describe larger areas use several filler fields together:

```
FIELDNAME = FILLER,,A256,A256,$
FIELDNAME = FILLER,,A200,A200,$
```

We recommend including file and segment attributes, even for simple files, to complete your documentation. The example below shows the Master File for the library file with file and segment declarations added.

```
FILENAME = LIBRARY1, SUFFIX = FIX,$
SEGNAME = BOOKS, SEGTYPE = S0,$
 FIELDNAME = PUBNO  ,ALIAS = PN    ,USAGE = A10   ,ACTUAL = A10    ,$
 FIELDNAME = AUTHOR ,ALIAS = AT    ,USAGE = A25   ,ACTUAL = A25    ,$
 FIELDNAME = TITLE  ,ALIAS = TL    ,USAGE = A50   ,ACTUAL = A50    ,$
 FIELDNAME = BINDING,ALIAS = BI    ,USAGE = A1    ,ACTUAL = A1     ,$
 FIELDNAME = PRICE  ,ALIAS = PR    ,USAGE = D8.2N ,ACTUAL = D8     ,$
 FIELDNAME = SERIAL ,ALIAS = SN    ,USAGE = A15   ,ACTUAL = A15    ,$
 FIELDNAME = FILLER ,ALIAS = FILL1 ,USAGE = A150  ,ACTUAL = A150   ,$
```

## What Are Free-Format Data Files?

A common type of external structure is a comma-delimited sequential file. These files are a convenient way to maintain low volumes of data, since the fields in a record are separated from one another by commas rather than being padded with blanks or zeros to fixed field lengths. Comma-delimited files must be stored as physical sequential files.

The FOCUS report request language processes comma-delimited data files in the same way it processes fixed-format files. The same procedure is used to describe these files in a comma-delimited Master File. The only difference is that the file suffix is changed to “COM”, as shown:

```
FILENAME = filename, SUFFIX = COM,$
```

You can use the system editor to change values, add new records, and delete records. Since the number of data fields on a line is variable, depending on the presence or absence of fields and the actual length of the data values, a logical record may be one or several lines. Hence, you need to use a terminator character to signal the end of the logical record. This is a dollar sign following the last comma (,$). A section of comma-delimited data might look like this:

```
PUBNO=1352334556, AUTHOR='Eliot, George',
TITLE='The Mill on the Floss', BINDING=H,$
```

The order in which the data values are described in the Master File plays an important role in comma-delimited files. If the data values are typed in their natural order, then only commas between the values are necessary. If a value is out of its natural order, then it is identified by its name or alias and an equal sign preceding it, for example, AUTHOR= ‘Eliot, George’.

## Rules for Maintaining Free-Format Data Files

If a logical record contains every data field, it will contain the same number of commas as delimiters as there are data fields. It will also have a dollar sign following the last comma, signaling the end of the logical record. Thus, a logical record containing ten data fields will contain ten commas as delimiters, plus a dollar sign record terminator.

A logical record may occupy as many lines in the data file as is necessary to contain the data. A record terminator (,$) must follow the last physical field.

Each record need not contain every data field, however. The identity of a data field that might be out of sequence can be provided in one of three different ways:

- You can use the field name, followed by an equal sign and the data value.
- You can use the field’s alias, followed by an equal sign and the data value.
- You can use the shortest unique truncation of the field’s name or alias, followed by an equal sign and the data value.

Thus, the following statements are all equivalent.

```
BI=H, PRICE=17.95,$
BI=H, PR=17.95,$
BI=H, P=17.95,$
```

# Standard Master File Attributes

Most standard Master File attributes—those described in Chapter 2, *Describing Data Sources*, Chapter 3, *Describing Groups of Fields*, and Chapter 4, *Describing Individual Fields*—are used with sequential data files in the standard way. The only exceptions is:

- **SEGTYPE.** The SEGTYPE attribute is not used with free-format data files.

  The SEGTYPE value for fixed-format data files defaults to S0. However, if you use keyed retrieval, the SEGTYPE value depends on the number of keys. See Chapter 7, *Describing FOCUS Databases*, for a description of the SEGTYPE attribute. For a description of keyed retrieval from fixed format data files, see the *Creating Reports* manual.

- **ACTUAL.** The ACTUAL values for sequential data files are described in Chapter 4, *Describing Individual Fields* in the section *The Stored Data Type: ACTUAL*.

Note that file and segment declarations are optional for simple sequential data files that you will not join. However, they are recommended to make the file description self-documenting, and to give you the option of joining the data file in the future.

# Describing Multiply-Occurring Fields in a Free-Format File

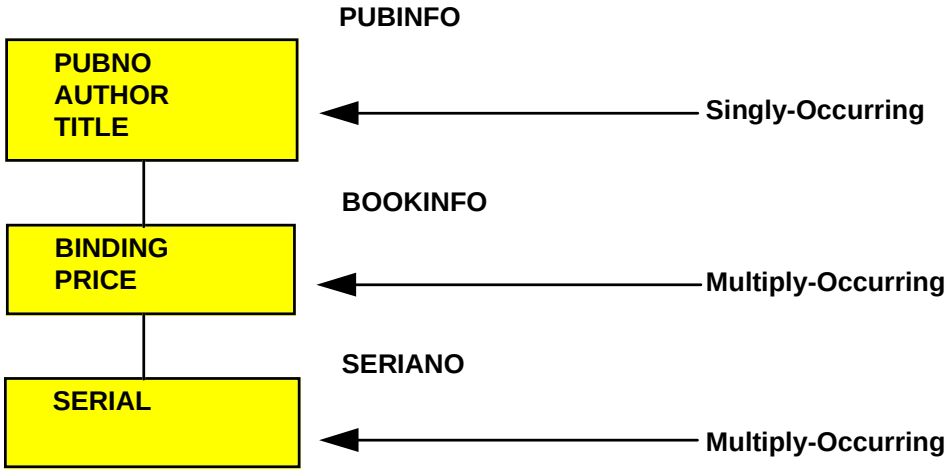
Since any data field not explicitly referred to in a logical record continues to have the same value it had the last time a value was assigned, up until the point a new data value is entered, a free-format sequential data file can resemble a hierarchical structure. The parent information need be entered only once, and it will carry over for each descendant segment.

### *Example* Multiply-Occurring Fields in a Free-Format File

Consider our example of a library file. The information for two copies of *The Sun Also Rises*, one hardcover and one paperback, can be entered as follows:

```
PUBNO=1234567890, AUTHOR='Hemingway, Ernest',
TITLE='The Sun Also Rises',
 BI=H,PR=17.95, $
 BI=S,PR=5.25, $
```

There are two values for binding and price, which both correspond to the same publisher's number, author, and title. In the FOCUS Master File, the information that occurs only once—the publisher's number, author, and title—is placed in one segment and the information that occurs several times in relation to this information is placed in a descendant segment. Similarly, information that occurs several times in relation to the descendant segment, such as an individual serial number for each copy of the book, is placed in a segment that is a descendant of the first descendant segment, as shown in the following diagram:




You describe this file with the following file description:

```
FILENAME = LIBRARY4, SUFFIX = COM, $

SEGNAME = PUBINFO, SEGTYPE=S0, $
    FIELDNAME = PUBNO,     ALIAS = PN,     USAGE = A10,      ACTUAL = A10,   $
    FIELDNAME = AUTHOR,    ALIAS = AT,     USAGE = A25,      ACTUAL = A25,   $
    FIELDNAME = TITLE,     ALIAS = TL,     USAGE = A50,      ACTUAL = A50,   $

SEGNAME = BOOKINFO, PARENT = PUBINFO,  SEGTYPE=S0, $
    FIELDNAME = BINDING,   ALIAS = BI,     USAGE = A1,       ACTUAL = A1,    $
    FIELDNAME = PRICE,     ALIAS = PR,     USAGE = D8.2N,    ACTUAL = D8,    $

SEGNAME = SERIANO,  PARENT = BOOKINFO, SEGTYPE=S0, $
    FIELDNAME = SERIAL,    ALIAS = SN,     USAGE = A15,      ACTUAL = A15,   $
```

Note that each segment other than the first has a PARENT attribute. You use the PARENT attribute to signal that you are describing a hierarchical structure.

# Describing Multiply-Occurring Fields in a Fixed-Format File

Fixed-format sequential data files can have repeating fields. Consider the following data structure:

| | | | | | |
|---|---|---|---|---|---|
| A | B | C1 | C2 | C1 | C2 |

Fields C1 and C2 repeat within this data record. C1 has an initial value, as does C2. C1 then provides a second value for that field, as does C2. Thus, there are two values for fields C1 and C2 for every one value for fields A and B.

The number of times C1 and C2 occur does not have to be fixed. It can depend on the value of a counter field. Suppose field B is this counter field. In the case shown above, the value of field B is 2, since C1 and C2 occur twice. The value of field B in the next record can be 7, 1, 0, or any other number you choose and fields C1 and C2 will occur that number of times.

The number of times fields C1 and C2 occur can also be variable. In this case, everything after fields A and B is assumed to be a series of C1s and C2s, alternating to the end of the record.

You describe these multiply-occurring fields by placing them in a separate segment. Fields A and B are placed in the root segment. Fields C1 and C2, which occur multiply in relation to A and B, are placed in a descendant segment. You use an additional segment attribute, the OCCURS attribute, to specify that these segments represent multiply-occurring fields. In certain cases, you may also need a second attribute, called the POSITION attribute.

## The OCCURS Attribute

The OCCURS attribute is an optional segment attribute used to describe records containing repeating fields or groups of fields. You define such records to FOCUS by describing the singly-occurring fields in one segment and the multiply-occurring fields in a descendant segment. The OCCURS attribute appears in the declaration for the descendant segment.

You can have several sets of repeating fields in your data structure. You describe each of these sets of fields as a separate segment in your file description. Sets of repeating fields can be divided into two basic types: parallel and nested.

## *Syntax* How to Specify Repeating Fields

The syntax of the OCCURS attribute is

`OCCURS = occurstype`

Valid values are:

*n*
: Is an integer value showing the number of occurrences (from 1 to 4095).

*fieldname*
: Names an integer field in the parent segment that is a counter containing the number of occurrences of the descendant segment.

`VARIABLE`
: Indicates that the number of occurrences varies from record to record. The number of occurrences is computed from the record length (that is, if the field lengths for the segment add up to 40, and 120 characters are read in, it means there are three occurrences).

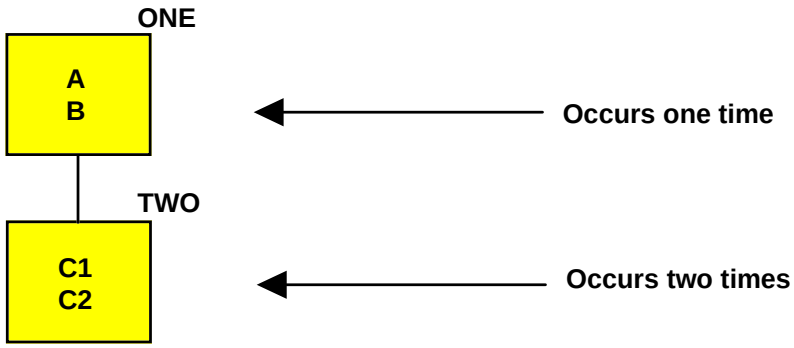
When different types of records are combined in one data file, each record type can contain only one segment defined as OCCURS=VARIABLE. It may have OCCURS descendants (if it contains a nested group), but it may not be followed by any other segment with the same parent—that is, there can be no other segments to its right in the hierarchical data structure. This restriction is necessary to ensure that data in the record is interpreted unambiguously by FOCUS.

*Example* **Using the OCCURS Attribute**

You place the OCCURS attribute in your segment declaration after the PARENT attribute. Consider the following simple data structure:

| | | | | | |
|---|---|---|---|---|---|
| A | B | C1 | C2 | C1 | C2 |

You have two occurrences of fields C1 and C2 for every one occurrence of fields A and B. Thus, to describe this data file, you place fields A and B in the root segment, and fields C1 and C2 in a descendant segment, as shown here:




You describe this data file with the following file description:

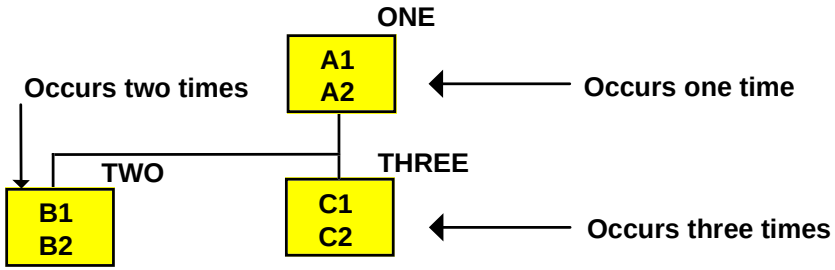
```
FILENAME = EXAMPLE1, SUFFIX = FIX, $

    SEGNAME = ONE, SEGTYPE=S0, $
    FIELDNAME = A,        ALIAS= ,     USAGE = A2,  ACTUAL = A2,            $
    FIELDNAME = B,        ALIAS= ,     USAGE = A1,  ACTUAL = A1,            $

    SEGNAME = TWO, PARENT = ONE, OCCURS = 2, SEGTYPE=S0, $
    FIELDNAME = C1,       ALIAS= ,     USAGE = I4,  ACTUAL = I2,            $
    FIELDNAME = C2,       ALIAS= ,     USAGE = I4,  ACTUAL = I2,            $
```

## Parallel Sets of OCCURS Segments

Parallel sets of repeating fields are those that have nothing to do with one another (that is, they have no parent-child or logical relationship). Consider the following data structure:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A1 | A2 | B1 | B2 | B1 | B2 | C1 | C2 | C1 | C2 | C1 | C2 |

In this example, fields B1 and B2 and fields C1 and C2 repeat within the record. The number of times that fields B1 and B2 occur has nothing to do with the number of times fields C1 and C2 occur. Fields B1 and B2 and fields C1 and C2 are parallel sets of repeating fields. They should be described in the file description as children of the same parent, the segment that contains fields A1 and A2. The following data structure reflects their relationship:

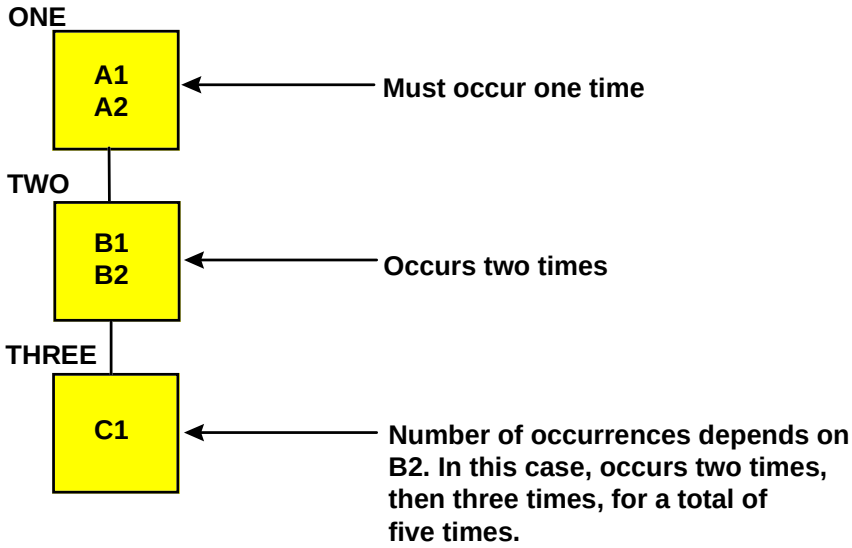


## Nested Sets of OCCURS Segments

Nested sets of repeating fields are those whose occurrence in some way depends on one another. Consider the following data structure:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| A1 | A2 | B1 | B2 | C1 | C1 | B1 | B2 | C1 | C1 | C1 |

In this example, field C1 only occurs after fields B1 and B2 occur once. It occurs varying numbers of times recorded by a counter field, B2. The one thing we know about this field is you will not have a set of occurrences of C1 without it being preceded by an occurrence of fields B1 and B2. Fields B1, B2, and C1 are a nested set of repeating fields. They can be represented by the following data structure:



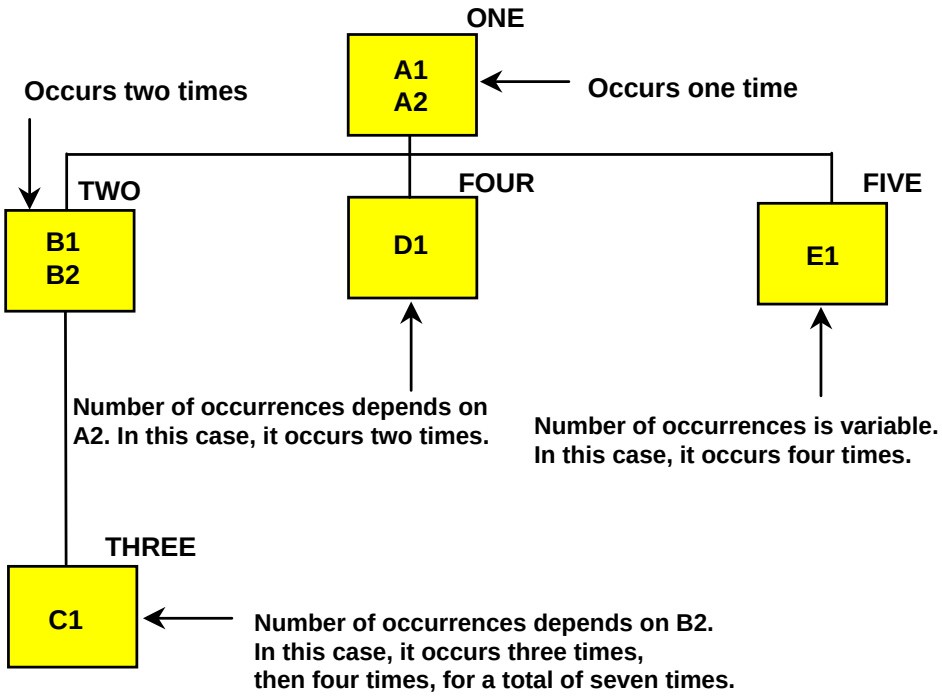

Since field C1 repeats with relation to fields B1 and B2, which repeat in relation to fields A1 and A2, field C1 is described as a separate, descendant segment of Segment Two, which is in turn a descendant of Segment One.

## Parallel and Nested Sets of OCCURS Statements

The following data structure contains both nested and parallel sets of repeating fields.

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A1 | A2 | B1 | B2 | C1 | C1 | C1 | B1 | B2 | C1 | C1 | C1 | C1 | D1 | D1 | E1 | E1 | E1 | E1 |

It produces the following data structure:



You describe this data file with the following file description. Notice that the assignment of the PARENT attributes shows you how the occurrences are nested.

```
FILENAME = EXAMPLE3, SUFFIX = FIX,$

    SEGNAME = ONE,   SEGTYPE=S0,$
    FIELDNAME = A1   ,ALIAS=   ,ACTUAL = A1   ,USAGE = A1    ,$
    FIELDNAME = A2   ,ALIAS=   ,ACTUAL = I1   ,USAGE = I1    ,$

    SEGNAME = TWO,   SEGTYPE=S0, PARENT = ONE, OCCURS = 2    ,$
    FIELDNAME = B1   ,ALIAS=   ,ACTUAL = A15  ,USAGE = A15   ,$
    FIELDNAME = B2   ,ALIAS=   ,ACTUAL = I1   ,USAGE = I1    ,$

    SEGNAME = THREE, SEGTYPE=S0, PARENT = TWO, OCCURS = B2   ,$
    FIELDNAME = C1   ,ALIAS=   ,ACTUAL = A25  ,USAGE = A25   ,$

    SEGNAME = FOUR,  SEGTYPE=S0, PARENT = ONE, OCCURS = A2   ,$
    FIELDNAME = D1   ,ALIAS=   ,ACTUAL = A15  ,USAGE = A15   ,$

    SEGNAME = FIVE,  SEGTYPE=S0, PARENT = ONE, OCCURS = VARIABLE,$
    FIELDNAME = E1   ,ALIAS=   ,ACTUAL = A5   ,USAGE = A5    ,$
```

**Note:**

- Segments ONE, TWO, and THREE represent a nested group of repeating segments. Fields B1 and B2 occur a fixed number of times, so OCCURS equals 2. Field C1 occurs a certain number of times within each occurrence of fields B1 and B2. The number of times C1 occurs is determined by the value of field B2, which is a counter. In this case, its value is 3 for the first occurrence of Segment TWO and 4 for the second occurrence.
- Segments FOUR and FIVE consist of fields that repeat independently within the parent segment. They have no relationship to each other or to Segment TWO except their common parent, so they represent a parallel group of repeating segments.
- As in the case of Segment THREE, the number of times Segment FOUR occurs is determined by a counter in its parent, A2. In this case, the value of A2 is two.
- The number of times Segment FIVE occurs is VARIABLE. This means that FOCUS will read all the rest of the fields in the record (all those to the right of the first occurrence of E1) as recurrences of field E1. To ensure that data in the record is interpreted unambiguously by FOCUS, a segment defined as OCCURS=VARIABLE must be at the end of the record. In a data structure diagram, it will be the right-most segment. Note that there can be only one segment defined as OCCURS=VARIABLE for each parent.

# The POSITION Attribute

The POSITION attribute is an optional attribute used to describe a structure in which multiply-occurring fields with an established number of occurrences are located in the middle of the record. You describe the data file to FOCUS as a hierarchical structure, made up of a parent segment and at least one child segment that contains the multiply-occurring fields. The parent segment is made up of whatever singly-occurring fields are in the record, as well as an alphanumeric field that appears where the multiply-occurring fields appear in the record. The alphanumeric field is a dummy field that is the exact length of the combined multiply-occurring fields. For example, if you have four occurrences of an eight-character field, the length of the dummy field in the parent segment will be 32 characters.

## *Syntax* How to Specify the Position of Repeating Fields

The POSITION attribute is described in the child segment. It gives the name of the dummy field in the parent segment that specifies the beginning of the multiply-occurring fields. The syntax of the POSITION attribute is

```
POSITION = fieldname
```

where:

*fieldname*
Is the name of the field in the parent segment that defines the overall length and starting position of the multiple field occurrences.

## *Example* Specifying the Position of Repeating Fields

Consider the following data structure:

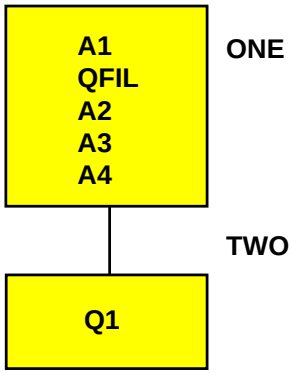
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A1 | Q1 | Q1 | Q1 | Q1 | A2 | A3 | A4 |

In this example, field Q1 repeats four times in the middle of the record. When you describe this structure to FOCUS, you specify a dummy field that occupies the position of the four Q1 fields in the record. You then assign the actual Q1 fields to a descendant, multiply-occurring segment. The POSITION attribute, specified in the descendant segment, gives the name of the dummy field in the parent segment.

You would use the following Master File to describe this structure:

```
FILENAME = EXAMPLE3, SUFFIX = FIX,$

SEGNAME = ONE, SEGTYPE=S0,$
     FIELDNAME = A1     ,ALIAS=       ,USAGE = A14      ,ACTUAL = A14    ,$
     FIELDNAME = QFIL   ,ALIAS=       ,USAGE = A32      ,ACTUAL = A32    ,$
     FIELDNAME = A2     ,ALIAS=       ,USAGE = I2       ,ACTUAL = I2     ,$
     FIELDNAME = A3     ,ALIAS=       ,USAGE = A10      ,ACTUAL = A10    ,$
     FIELDNAME = A4     ,ALIAS=       ,USAGE = A15      ,ACTUAL = A15    ,$

SEGNAME = TWO, SEGTYPE=S0, PARENT = ONE, POSITION = QFIL, OCCURS = 4 ,$
     FIELDNAME = Q1     ,ALIAS=       ,USAGE = D8       ,ACTUAL = D8     ,$
```

This produces the following structure:




If the total length of the multiply-occurring of the field(s) is longer than 256, you can use a filler field after the dummy field to make up the remaining length. This is required because the format of an alphanumeric field cannot exceed 256 bytes.

Notice that this structure will only work if you have a fixed number of occurrences of the repeating field. This means the OCCURS attribute of the descendant segment must be of the type OCCURS=n. OCCURS=field name or OCCURS=VARIABLE will not work.

# Specifying the ORDER Field

In an OCCURS segment, the order of the data may be significant. For example, the values may represent monthly or quarterly data, but the record itself may not explicitly specify the month or quarter to which the data applies.

If you wish to learn the sequence number of each field, you may include an additional dummy field in any OCCURS segment. FOCUS automatically supplies a value for it that defines the sequence number of each repeating group.

*Syntax* **How to Specify the Sequence of Repeating Fields**

The syntax rules for an ORDER field are:

- It must be the last field described in an OCCURS segment.
- The field name is arbitrary.
- The ALIAS is ORDER.
- The USAGE is In, with any appropriate edit options.
- The ACTUAL is I4.

For example:

```
FIELD = ACT_MONTH, ALIAS = ORDER, USAGE = I2MT, ACTUAL = I4, $
```

Order values are 1, 2, 3, and so on, within each occurrence of the segment. The value is assigned prior to any selection tests that might accept or reject the record, and so it can be used in a selection test.

For example, to obtain data for only the month of June, type:

```
SUM AMOUNT...
WHERE ACT_MONTH IS 6
```

The ORDER field is a virtual field used internally by FOCUS. It does not alter the logical record length (LRECL) of the file being accessed.

# Redefining Fields in Non-FOCUS Data Sources

Support is provided for redefining record fields in non-FOCUS data sources. This allows a field to be described with an alternate layout.

Within the Master File, the redefined fields must be described in a separate unique segment (SEGTYPE=U) using the POSITION=fieldname and OCCURS=1 attributes.

## *Syntax* How to Redefine a Field

The syntax for a redefinition segment is:

```
SEGNAME = segname, SEGTYPE = U, PARENT = parentseg,
OCCURS = 1, POSITION = fieldname,$
```

where:

*segname*
Is the name of the segment.

*parentseg*
Is the name of the parent segment.

*fieldname*
Is the name of the field being redefined.

A one-to-one relationship is established between the parent record and the redefined segment.

## *Example* Redefining a VSAM Structure

The following example illustrates redefinition of the VSAM structure described in the COBOL file description where the COBOL FD is:

```
01 ALLFIELDS
  02 FLD1   PIC X(4)         -  this describes alpha/numeric data
  02 FLD2   PIC X(4)         -  this describes numeric data
  02 RFLD1  PIC 9(5)V99 COMP-3 REDEFINES FLD2
  02 FLD3   PIC X(8)         -  this describes alpha/numeric data

FILE = REDEF, SUFFIX = VSAM,$
 SEGNAME = ONE, SEGTYPE = S0,$
    GROUP = RKEY, ALIAS = KEY,USAGE = A4    ,ACTUAL = A4   ,$
      FIELDNAME = FLD1,,      USAGE = A4    ,ACTUAL = A4   ,$
      FIELDNAME = FLD2,,      USAGE = A4    ,ACTUAL = A4   ,$
      FIELDNAME = FLD3,,      USAGE = A8    ,ACTUAL = A8   ,$
 SEGNAME = TWO, SEGTYPE = U, POSITION = FLD2, OCCURS = 1, PARENT = ONE ,$
      FIELDNAME = RFLD1,,     USAGE = P8.2  ,ACTUAL = Z4   ,$
```

The redefined fields can have any user-defined name. ALIAS names for redefined field are not required.

Use of the unique segment with redefined fields helps avoid problems with multipath reporting.

### *Reference* Special Considerations for Redefining Fields

- Redefinition is a read-only feature and is used only for presenting an alternate view of the data. It is not used for changing the format of the data.
- A field that is being redefined must be equal in length to the field that it is redefining (same actual length).
- For non-alphanumeric fields, you must know your data. Attempts to print numeric fields that contain alpha data will produce data exceptions or errors converting values.
- More than one field can be redefined in a segment.

Redefines are only supported for IDMS, IMS, VSAM, DB2, and FIX files.

## Extra-Large Record Length Support

If the Master File describes a data file with OCCURS segments, and if the longest single record in the file is larger than 16K bytes, it is necessary to specify a larger record size to FOCUS in advance.

To define the maximum record length, type:

```
SET MAXLRECL = nnnn
```

where *nnnn* is up to 32000.

For example, SET MAXLRECL=12000 allows handling of records that are 12000 bytes long. Once you have entered the SET MAXLRECL command, you can obtain the current value of the MAXLRECL parameter by using the ? SET command.

If the actual record length is longer than specified, retrieval is halted and the actual record length is displayed in hexadecimal notation.

# Describing Multiple Record Types

Sequential, fixed-format data files can contain more than one type of record. When they do, they can be structured in one of two possible ways.

- A logical relationship may exist between the various record types, with a record of one type being followed by one or more records containing detailed information about the first record.

  If a logical relationship exists between the various record types, with a parent record of one type followed by one or more child records containing detail information about the parent, you describe the structure to FOCUS by defining the parent as the root and the detail segments as descendants.

- The records have no meaningful logical relationship, and records of varying RECTYPEs exist independently of each other in the file.

  If the records have no meaningful logical relationship, you have to provide some means whereby FOCUS can interpret what type of record has been read in. You do this by creating a dummy root segment for the records.

In order to describe sequential data files with several types of records, regardless of whether they are logically related, use the PARENT segment attribute and the RECTYPE field attribute. Any complex sequential data file is described as a multi-segment structure to FOCUS.

# Describing the RECTYPE Field

When a file contains multiple record types, there must be a field in the records themselves (sometimes called a “flag”) that can be used to differentiate between them. You can find information on this field in your existing description of the file (for example, a COBOL FD statement). This field must appear in the same physical location of each record. For example, columns 79 and 80 could contain a different two-digit code for each unique record type. In FOCUS, you describe this identifying field with a field name of RECTYPE.

## *Syntax* How to Specify a ‘Flag’ Field

The RECTYPE field must fall in the same physical location of each record in the data file or the record will be ignored by FOCUS. The syntax to describe the RECTYPE field is

```
FIELDNAME = RECTYPE, ALIAS = value, USAGE = format, ACTUAL = format ,$
```

where:

*value*
Is the record type in alphanumeric format.

*format*
Is the data type of the field.

`A`*n* (where *n* is 1-256) indicates character data, including letters, digits, and other ASCII characters.

`I`*n* indicates binary integers:

I1 = single-byte binary integer.
I2 = half-word binary integer (2 bytes).
I4 = full-word binary integer (4 bytes).

`P`*n* (where *n* is 1-16) indicates packed decimal internal format. *n* is the number of bytes, each of which contains two digits, except for the last byte which contains a digit and the sign (+ or -). For example, P6 means 11 digits plus a sign.

If the field contains an assumed decimal point, represent the field with a USAGE format of Pm.n, where *m* is the total number of bytes, and *n* is the number of decimal places. Thus P6.1 means an eleven-digit number with one decimal place.

### *Example* Specifying the RECTYPE Field

FOCUS supports RECTYPE fields in packed and integer formats (formats P and I), in addition to RECTYPE fields in alphanumeric format.

For example, the following field description describes a one-byte packed RECTYPE field containing the value 1:

```
FIELD = RECTYPE, ALIAS = 1, USAGE = P1, ACTUAL = P1, $
```

This field description describes a three-byte alphanumeric RECTYPE field containing the value A34:

```
FIELD = RECTYPE, ALIAS = A34, USAGE = A3, ACTUAL = A3,$
```

## Using Generalized RECTYPES

If your fixed-format sequential file or VSAM file has multiple record types that share the same layout, you can specify a single “generalized” segment that describes all record types having the common layout. By using a generalized segment—also known as a generalized RECTYPE—instead of one segment per record type, you reduce the number of segments you need to describe in the Master File.

When you use a generalized RECTYPE segment, you identify RECTYPE values using the ACCEPT attribute. You can assign any value you wish to the ALIAS attribute.

### *Syntax* How to Specify Generalized RECTYPES

The syntax is

```
FIELDNAME = RECTYPE, ALIAS = alias, USAGE = format, ACTUAL = format,
ACCEPT = {list|range} ,$
```

where:

`RECTYPE`
: Is the required field name.

*`alias`*
: Is any valid alias specification. You can specify a unique name as the alias value for the RECTYPE field only if you use the ACCEPT attribute. The alias can then be used in a TABLE request as a display field, a sort field, or in selection tests using either WHERE or IF. **Note:** Since the field name, RECTYPE, may not be unique across segments, you should not use it in this way unless you qualify it. An alias is not required; you may leave it blank.

*list*

Is a list of one or more lines of specific RECTYPE values for records that have the same segment layout. The maximum number of characters allowed in the list is 255. Each item in the list must be separated by either a blank or the keyword OR. If the list contains embedded blanks or commas, it must be enclosed within single quotation marks. The list may contain a single RECTYPE value.

For example:

```
FIELDNAME = RECTYPE, ALIAS = TYPEABC, USAGE = A1,
ACTUAL = A1, ACCEPT = A OR B OR C, $
```

*range*

Is a range of one or more lines of RECTYPE values for records that have the same segment layout. The maximum number of characters allowed in the range is 255. If the range contains embedded blanks or commas, it must be enclosed within single quotation marks.

To specify a range of values, include the lowest value, the keyword TO, and the highest value, in that order. For example:

```
FIELDNAME = RECTYPE, ALIAS = ACCTREC, USAGE = P3,
ACTUAL = P2, ACCEPT = 100 TO 200, $
```

## *Example* Using Generalized RECTYPES

To illustrate the use of the generalized RECTYPE capability in VSAM Master Files, consider the following record layouts in the DOC file. Record type DN is the root segment and contains the document number and title. Record types M, I, and C contain information about manuals, installation guides, and course guides, respectively. Notice that record types M and I have the same layout.

Record Type DN:

```
---KEY---
+-------------------------------------------------------------------------------
+
DOCID   FILLER            RECTYPE      TITLE
+-------------------------------------------------------------------------------
+
```

Record Type M:

```
--------KEY--------
+-------------------------------------------------------------------------------
+
MDOCID  MDATE             RECTYPE      MRELEASE                 MPAGES
    FILLER
+-------------------------------------------------------------------------------
+
```

Record Type I:

```
--------KEY--------
+-------------------------------------------------------------------------------
+
IDOCID  IDATE             RECTYPE      IRELEASE                 IPAGES
    FILLER
+-------------------------------------------------------------------------------
+
```

Record Type C:

```
--------KEY--------
+-------------------------------------------------------------------------------
+
CRSEDOC CDATE             RECTYPE      COURSENUM    LEVEL        CPAGES
    FILLER
+-------------------------------------------------------------------------------
+
```

Without the ACCEPT attribute, each of the four record types must be described as separate segments in the Master File. In particular, a unique set of field names must be provided for record type M and for record type I, although they have the same layout.

The generalized RECTYPE capability enables you to code just one set of field names that will apply to the record layout for both record type M and record type I. The ACCEPT attribute can be used for any RECTYPE specification, even when there is only one acceptable value.

```
FILENAME=DOC2, SUFFIX=VSAM,$
SEGNAME=ROOT, SEGTYPE=SO,$
 GROUP=DOCNUM, ALIAS=KEY, A5, A5,$
  FIELD=DOCID,     ALIAS=SEQNUM,    A5,  A5, $
 FIELD=FILLER,     ALIAS,=          A5,  A5, $
 FIELD=RECTYPE,    ALIAS=DOCRECORD, A3,  A3, ACCEPT =DN,$
 FIELD=TITLE,      ALIAS=,          A18, A18, $
SEGNAME=MANUALS, PARENT=ROOT, SEGTYPE=SO,$
 GROUP=MDOCNUM,ALIAS=KEY, A10, A10,$
  FIELD=MDOCID,    ALIAS=MSEQNUM,   A5,  A5, $
  FIELD=MDATE,     ALIAS=MPUBDATE,  A5,  A5, $
 FIELD=RECTYPE,    ALIAS=MANUAL,    A3,  A3, ACCEPT = M OR I,$
 FIELD=MRELEASE,   ALIAS=,          A7,  A7, $
 FIELD=MPAGES,     ALIAS=,          I5,  A5, $
 FIELD=FILLER,     ALIAS=,          A6,  A6, $
SEGNAME=COURSES, PARENT=ROOT, SEGTYPE=SO,$
 GROUP=CRSEDOC, ALIAS=KEY, A10, A10,$
  FIELD=CDOCID,    ALIAS=CSEQNUM,   A5,  A5, $
  FIELD=CDATE,     ALIAS=CPUBDATE,  A5,  A5, $
 FIELD=RECTYPE,    ALIAS=COURSE,    A3,  A3, ACCEPT = C,$
 FIELD=COURSENUM,  ALIAS=CNUM,      A4,  A4, $
 FIELD=LEVEL,      ALIAS=,          A2,  A2, $
 FIELD=CPAGES,     ALIAS=,          I5,  A5, $
 FIELD=FILLER,     ALIAS=,          A7,  A7, $
```

# Using the ALIAS in Report Requests

You can include an alias for the RECTYPE field if you use the ACCEPT attribute to specify one or more RECTYPE values in the Master File. This enables you to use the alias in a report request as a display field, as a sort field, or in selection tests using either WHERE or IF.

## *Example* Using RECTYPE Values in a Display Command

You can display the RECTYPE values by including the alias as a display field. In this example, the alias MANUAL displays the RECTYPE values M and I:

```
TABLE FILE DOC
PRINT MANUAL MRELEASE MPAGES
BY DOCID BY TITLE BY MDATE
END
```

```
PAGE 1

DOCID    TITLE                MDATE   RECTYPE   MRELEASE   MPAGES
-----    -----                -----   -------   --------   ------
40001    FOCUS USERS MANUAL   8601    M         5.0        1800
                              8708    M         5.5        2000
40057    MVS INSTALL GUIDE    8806    I         5.5.3        66
                              8808    I         5.5.4        66
40114    CMS INSTALL GUIDE    8806    I         5.5.3        58
                              8808    I         5.5.4        58
```

## *Example* Using RECTYPE Values in a WHERE Test

You can use the alias in a WHERE test to display a subset of records.

```
TABLE FILE DOC
PRINT MANUAL MRELEASE MPAGES
BY DOCID BY TITLE BY MDATE
WHERE MANUAL EQ 'I'
END
```

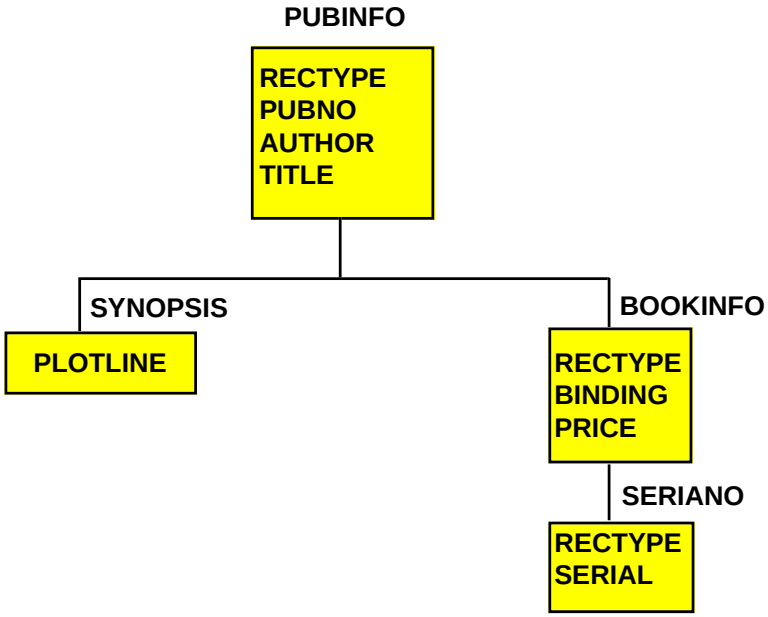
```
PAGE 1

DOCID    TITLE                MDATE   RECTYPE   MRELEASE   MPAGES
-----    -----                -----   -------   --------   ------
40057    MVS INSTALL  GUIDE   8806    I         5.5.3          66
                              8808    I         5.5.4          66
40114    CMS INSTALL GUIDE    8806    I         5.5.3          58
                              8808    I         5.5.4          58
```

# Describing Logically Related Records

Parent-child relationships determine how FOCUS interprets the contents of a data file. In a file with logically related records, a parent-child-type relationship has already been created. In order to describe such a file as a FOCUS structure, the first record type should be assigned to the root segment, while the detail records that follow it should be assigned to its various descendant segments, each with their own record type.

***Example*** **Describing Logically Related Records**

The following diagram shows a more complex version of the library file discussed previously.




Information that is common to all copies of a given book (the identifying number, the author's name, and its title) has the same record type. They are all assigned to the root segment in the FOCUS description. The synopsis is common to all copies of a given book, but, in this data file, it has been described as a series of repeating fields of ten characters each in order to save space.

The synopsis is assigned to its own subordinate segment with an attribute of OCCURS=VARIABLE in the Master File. Although there are segments that are shown in the diagram to the right of the OCCURS=VARIABLE segment, the OCCURS=VARIABLE segment is the right-most segment within its own record type. Only segments with a RECTYPE that is different from the OCCURS=VARIABLE segment can appear to its right in the FOCUS structure. Note also that the OCCURS=VARIABLE segment does not have a RECTYPE. This is because the OCCURS=VARIABLE segment is part of the same record as its parent segment.

Binding and price can vary among copies of a given title. For instance, the library may have two different versions of *Pamela*, one a paperback costing $7.95, the other a hardcover costing $15.50. These two fields are of a second record type, and are assigned to a descendant segment in the Master File.

Finally, every copy of the book in the library will have its own identifying serial number, which will be described in a field of record type S. In the Master File, this information is assigned to a segment that is a child of the segment containing the binding and price information.

You would use the following Master File to describe this data file:

```
FILENAME = LIBRARY2, SUFFIX = FIX,$

SEGNAME = PUBINFO, SEGTYPE = S0,$
 FIELDNAME = RECTYPE  ,ALIAS = P   ,USAGE = A1    ,ACTUAL = A1   ,$
 FIELDNAME = PUBNO    ,ALIAS = PN  ,USAGE = A10   ,ACTUAL = A10  ,$
 FIELDNAME = AUTHOR   ,ALIAS = AT  ,USAGE = A25   ,ACTUAL = A25  ,$
 FIELDNAME = TITLE    ,ALIAS = TL  ,USAGE = A50   ,ACTUAL = A50  ,$

SEGNAME = SYNOPSIS , PARENT = PUBINFO, OCCURS = VARIABLE, SEGTYPE = S0,$
 FIELDNAME = PLOTLINE ,ALIAS = PLOTL ,USAGE = A10 ,ACTUAL = A10  ,$

SEGNAME = BOOKINFO, PARENT = PUBINFO, SEGTYPE = S0,$
 FIELDNAME = RECTYPE  ,ALIAS = B   ,USAGE = A1    ,ACTUAL = A1   ,$
 FIELDNAME = BINDING  ,ALIAS = BI  ,USAGE = A1    ,ACTUAL = A1   ,$
 FIELDNAME = PRICE    ,ALIAS = PR  ,USAGE = D8.2N ,ACTUAL = D8   ,$

SEGNAME = SERIANO, PARENT = BOOKINFO, SEGTYPE = S0,$
 FIELDNAME = RECTYPE  ,ALIAS = S   ,USAGE = A1    ,ACTUAL = A1   ,$
 FIELDNAME = SERIAL   ,ALIAS = SN  ,USAGE = A15   ,ACTUAL = A15  ,$
```

Note that each segment, except the OCCURS segment, contains a field named RECTYPE and that the ALIAS for the field contains a unique value for each segment (P, B, and S). If FOCUS encounters a record in this file with a RECTYPE other than P, B, or S, the record will be ignored. The RECTYPE field must fall in the same physical location in each record.

## Order of Records in the Data File

Every parent record need not have descendants. You can specify how you want data in missing segment instances handled in your reports by using the SET command to change the ALL parameter. The SET command is described in the *Developing Applications* manual.

In the structure shown in the example in *Describing Logically Related Records*, on page 5-25, if the first record in the data file is not a PUBINFO record, the file is considered to be in error. Any information allotted to the SYNOPSIS segment will appear in the PUBINFO record. The next record may be a BOOKINFO or even another PUBINFO (in which case the first PUBINFO is assumed to have no descendants). Any SERIANO records are assumed to be descendants of the previous BOOKINFO record. If a SERIANO record follows a PUBINFO record with no intervening BOOKINFO, it is treated as an error since it has no parent.

# Describing Unrelated Records

Some files have records that are not logically related to each other. That is, records of varying RECTYPEs exist independently of each other in the data file. Thus, the sequence of records in the file may be arbitrary. To describe such files in FOCUS you define a dummy root segment. To describe files with unrelated records, you make the record types descendants of a dummy root segment. The following rules apply to the dummy root segment:

- The name of the root segment must be DUMMY.
- It must have only one field with an empty name and alias.
- Its USAGE and ACTUAL attributes must both be A1.

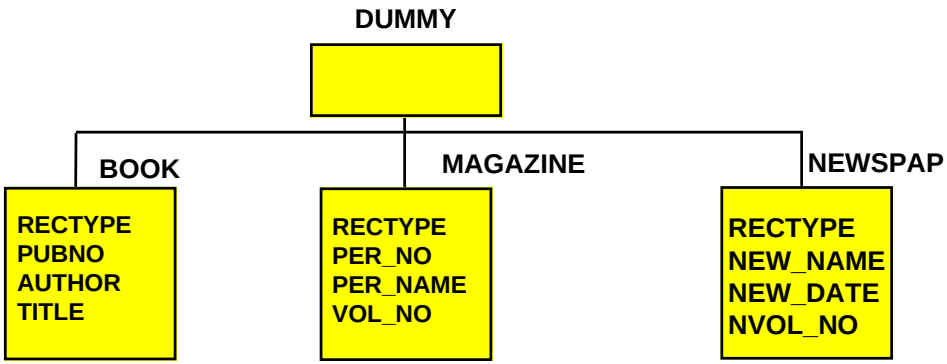
All of the other non-OCCURS segments must point to the dummy root as their parent. Except for the root, all non-OCCURS segments must have a RECTYPE and a PARENT attribute.

*Example* **Describing Unrelated Records Using a Dummy Root Segment**

For example, the library database has three types of records: book information, magazine information, and newspaper information. Since book information, magazine information, and newspaper information have nothing in common, these three record types cannot be described as parent records followed by detail records.

The file might look like this:


A structure such as the following could also describe this file:



The Master File for the structure in this example is:

```
FILENAME = LIBRARY3, SUFFIX = FIX,$

SEGMENT = DUMMY, SEGTYPE = S0,$
 FIELDNAME=           ,ALIAS=     ,USAGE = A1    ,ACTUAL = A1  ,$

SEGMENT = BOOK, PARENT = DUMMY, SEGTYPE = S0,$
 FIELDNAME = RECTYPE  ,ALIAS = B  ,USAGE = A1    ,ACTUAL = A1  ,$
 FIELDNAME = PUBNO    ,ALIAS = PN ,USAGE = A10   ,ACTUAL = A10 ,$
 FIELDNAME = AUTHOR   ,ALIAS = AT ,USAGE = A25   ,ACTUAL = A25 ,$
 FIELDNAME = TITLE    ,ALIAS = TL ,USAGE = A50   ,ACTUAL = A50 ,$
 FIELDNAME = BINDING  ,ALIAS = BI ,USAGE = A1    ,ACTUAL = A1  ,$
 FIELDNAME = PRICE    ,ALIAS = PR ,USAGE = D8.2N ,ACTUAL = D8  ,$
 FIELDNAME = SERIAL   ,ALIAS = SN ,USAGE = A15   ,ACTUAL = A15 ,$
 FIELDNAME = SYNOPSIS ,ALIAS = SY ,USAGE = A150  ,ACTUAL = A150,$

SEGMENT = MAGAZINE, PARENT = DUMMY, SEGTYPE = S0,$
 FIELDNAME = RECTYPE  ,ALIAS = M  ,USAGE = A1    ,ACTUAL = A1  ,$
 FIELDNAME = PER_NO   ,ALIAS = PN ,USAGE = A10   ,ACTUAL = A10 ,$
 FIELDNAME = PER_NAME ,ALIAS = NA ,USAGE = A50   ,ACTUAL = A50 ,$
 FIELDNAME = VOL_NO   ,ALIAS = VN ,USAGE = I2    ,ACTUAL = I2  ,$
 FIELDNAME = ISSUE_NO ,ALIAS = IN ,USAGE = I2    ,ACTUAL = I2  ,$
 FIELDNAME = PER_DATE ,ALIAS = DT ,USAGE = I6MDY ,ACTUAL = I6  ,$

SEGMENT = NEWSPAP, PARENT = DUMMY, SEGTYPE = S0,$
 FIELDNAME = RECTYPE  ,ALIAS = N  ,USAGE = A1    ,ACTUAL = A1  ,$
 FIELDNAME = NEW_NAME ,ALIAS = NN ,USAGE = A50   ,ACTUAL = A50 ,$
 FIELDNAME = NEW_DATE ,ALIAS = ND ,USAGE = I6MDY ,ACTUAL = I6  ,$
 FIELDNAME = NVOL_NO  ,ALIAS = NV ,USAGE = I2    ,ACTUAL = I2  ,$
 FIELDNAME = ISSUE    ,ALIAS = NI ,USAGE = I2    ,ACTUAL = I2  ,$
```

# Combining Multiply-Occurring Fields and Multiple Record Types

You can have two types of descendant segments in a single fixed-format sequential data file: descendant segments consisting of multiply-occurring fields, and additional descendant segments consisting of multiple record types.

In the data structure shown below, the first record—of type 01—contains several different sequences of repeating fields, all of which must be described to FOCUS as descendant segments possessing an OCCURS attribute. The data file also contains two separate records, of types 02 and 03, which contain information that is related to that in record type 01.

The relationships between the records of various types is expressed as parent-child relationships. The children that contain record types 02 and 03 will not have an OCCURS attribute. They are distinguished from their parent by the field declaration where field=RECTYPE.

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 01 | T1 | N1 | B1 | B2 | C1 | C1 | C1 | D1 | D1 | D1 | D1 | D1 | D1 | D1 | B1 | B2 | C1 | C1 | D1 | D1 | D1 | D1 | D1 | D1 | D1 |

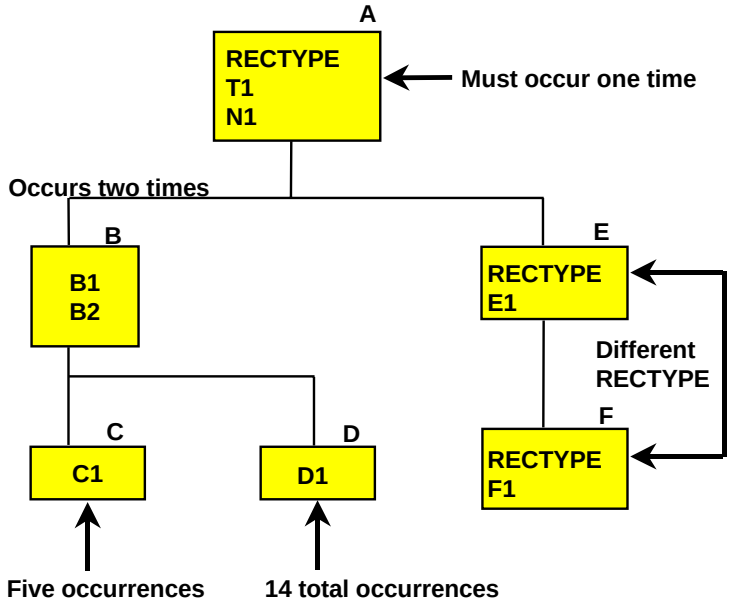
| | |
|---|---|
| 02 | E1 |

| | |
|---|---|
| 03 | F1 |

The file description for this file is:

```
FILENAME = EXAMPLE1, SUFFIX = FIX,$

SEGNAME = A, SEGTYPE=S0,$
 FIELDNAME = RECTYPE     ,ALIAS = 01    ,USAGE = A2    ,ACTUAL = A2      ,$
 FIELDNAME = T1          ,ALIAS =       ,USAGE = A2    ,ACTUAL = A1      ,$
 FIELDNAME = N1          ,ALIAS =       ,USAGE = A1    ,ACTUAL = A1      ,$

SEGNAME = B, PARENT = A, OCCURS = VARIABLE, SEGTYPE=S0,$
 FIELDNAME = B1          ,ALIAS =       ,USAGE = I2    ,ACTUAL = I2      ,$
 FIELDNAME = B2          ,ALIAS =       ,USAGE = I2    ,ACTUAL = I2      ,$

SEGNAME = C, PARENT = B, OCCURS = B1, SEGTYPE=S0,$
 FIELDNAME = C1          ,ALIAS =       ,USAGE = A1    ,ACTUAL = A1      ,$

SEGNAME = D, PARENT = B, OCCURS = 7, SEGTYPE=S0,$
 FIELDNAME = D1          ,ALIAS =       ,USAGE = A1    ,ACTUAL = A1      ,$

SEGNAME = E, PARENT = A, SEGTYPE=S0,$
 FIELDNAME = RECTYPE     ,ALIAS = 02    ,USAGE = A2    ,ACTUAL = A2      ,$
 FIELDNAME = E1          ,ALIAS =       ,USAGE = A1    ,ACTUAL = A1      ,$

SEGNAME = F, PARENT = E, SEGTYPE=S0,$
 FIELDNAME = RECTYPE     ,ALIAS = 03    ,USAGE = A2    ,ACTUAL = A2      ,$
 FIELDNAME = F1          ,ALIAS =       ,USAGE = A1    ,ACTUAL = A1      ,$
```

It produces the following data structure:




Segments A, B, C, and D all belong to the same record type. Segments E and F each are stored as separate record types.

**Note:**

- Segments A, E, and F are different records, related through their record types. The record type attribute consists of certain prescribed values and is stored in a fixed location in the records. FOCUS expects to read the records in a given order. If the first record does not have a RECTYPE of 01, the file is considered to be in error. The next record can have a RECTYPE of either 01 (in which case the first record is considered to have no descendants except the OCCURS descendants) or 02. A record with a RECTYPE of 03 can follow only a record with a RECTYPE of 02 (its parent).
- The OCCURS descendants all belong to the record whose RECTYPE is 01. (This is not a necessary condition; records of any type can have OCCURS descendants.) Note that the OCCURS=VARIABLE segment, Segment B, is the right-most segment within its own record type. If you look at the data structure shown above, the pattern that makes up Segment B and its descendants (the repetition of fields B1, B2, C1, and D1) extends from the first mention of fields B1 and B2 to the end of the record.
- Although fields C1 and D1 appear in separate segments, they are actually part of the repeating pattern that makes up the OCCURS=VARIABLE segment. Since they occur multiple times within Segment B, they are each assigned to their own descendant segment. The number of times field C1 occurs depends on the value of field B2. In the example, the first value of field B2 is 3, the second, 2. Field D1 occurs a fixed number of times, 7.

## Reading Complex Files With User-Written Procedures

There are three ways you can read non-FOCUS data sources with user-written procedures. All three are described in Appendix C, *User Exits for Non-FOCUS Data Sources*.

- You can invoke a user exit contained in the FOCSAM Interface and combine simple user written code with the Interface's logical functions.
- You can also write an independent user routine to provide records to the FOCUS report writer.

  The records, which can come from any source, are treated by FOCUS exactly as if they had come from a FOCUS database. The user routine must be coded as a subroutine in FORTRAN, COBOL, BAL, or PL/I, and passes data to the FOCUS calling program through arguments in the subroutine.

  The user program is loaded automatically by the report writer. It is identified by its file suffix, in the Master File. The suffix is of the form

  ```
  SUFFIX = program_name
  ```

  where *program_name* is the name of your user-written subroutine.

  Thus if your Master File contains

  ```
  FILE = ABC, SUFFIX = MYREAD
  ```

  FOCUS will load and call the program named MYREAD to obtain data whenever there is a TABLE, TABLEF, MATCH, or GRAPH command for file ABC.
- A decompression exit is also available for compressed VSAM files and flat files. This is called ZCOMP1. It uses SUFFIX=PRIVATE.

# 6 Describing ISAM and VSAM Files

**Topics:**

- Simple ISAM and Key-Sequenced VSAM Files
- Complex ISAM and Key-Sequenced VSAM Files

Previous discussions of techniques for describing fixed-format records assumed that the files were sequential (QSAM) or entry-sequenced VSAM files and contained no keys. With ISAM and VSAM files a new element is introduced, the “key” or “group key.” A “group key” consists of one or more fields used to identify the various record types in the file. In the FOCUS representation of such a file, each different record type is assigned its own segment.

**Note:** For ISAM and VSAM files, you must allocate (in MVS) or DLBL (in DOS/VSE and VM) the Master File name to the CLUSTER component of the file.

Also, FOCUS supports ESDS and KSDS files, not RRDS VSAM files. If you wish to retrieve data from RRDS VSAM files, you may code your own access routine using SUFFIX=PRIVATE. See Appendix C, *User Exits for Non-FOCUS Data Sources* for details. For information about updating VSAM files, see the *FOCUS VSAM Read/Write Interface User’s Manual*.

## Simple ISAM and Key-Sequenced VSAM Files

The standard FOCUS attributes for describing files, segments, and fields apply to ISAM or VSAM files, with the following conditions.

### File Suffix for ISAM or VSAM Files

The SUFFIX attribute in the file declaration for these files has the value ISAM or VSAM.

### Segment Name for ISAM or VSAM Files

The SEGNAME of the first or root segment in a FOCUS Master File for an ISAM or VSAM file must be ROOT. The remaining segments can have any valid segment name.

The only exception involves unrelated RECTYPES, where the first SEGNAME must be DUMMY.

All non-repeating data goes in the root segment. The remaining segments may have any valid name from one to eight characters.

Any segment except the root is the descendant, or child, of another segment. The PARENT attribute supplies the name of the segment which is the hierarchical parent or owner of the current segment. If no PARENT attribute appears, the default is the immediately preceding segment. The PARENT name may be one to eight characters.

### SEGTYPE for VSAM Files

The SEGTYPE attribute should be S0 for VSAM files. (For a general description of the SEGTYPE attribute, see Chapter 3, *Describing Groups of Fields*.)

## Describing Groups

The keys of an ISAM or VSAM file are defined in the SEGMENT declarations as GROUPs consisting of one or more fields. The GROUP declaration has the following syntax

```
GROUP = keyname, ALIAS = KEY, USAGE = Ann, ACTUAL = Ann ,$
```

where *keyname* can have up to 66 characters.

A single segment file may have only one keyfield, but it must still be described with a GROUP declaration. The group must have ALIAS=KEY.

Groups can also be assigned simply to provide convenient reference names for groups of fields. Suppose, for instance, that you have a series of three fields for an employee: last name; first name; and social security number. You use these three fields consistently to identify the employee. You can identify the three fields in your Master File as a GROUP named EMPINFO. Then, you can refer to these three linked fields as a single unit, called EMPINFO. When using the GROUP feature for non-keys, the parameter ALIAS= must still be used, but should not equal KEY.

For group fields you must supply both the USAGE and ACTUAL formats in alphanumeric format. The length must be exactly the sum of the subordinate field lengths.

If the fields that make up a group key are not alphanumeric fields, the format of the group key is still alphanumeric, but its length is determined differently. The ACTUAL length is still the sum of the subordinate field lengths. The USAGE format, however, is the sum of the internal storage lengths of the subordinate fields. You determine these internal storage lengths as follows:

- Fields of type I have a value of 4.
- Fields of type F have a value of 4.
- Fields that are 8 bytes have a USAGE of P15, P16, P17.n (sign and decimal for a total of 15 digits). Fields that are 16 bytes will have a USAGE of P17 or larger.
- Fields of type D have a value of 8.
- Alphanumeric fields have a value equal to the number of characters they contain as their field length.

Consider the following example:

```
GROUP = A, ALIAS = KEY, USAGE = A14, ACTUAL = A8  ,$
 FIELDNAME = F1, ALIAS = F1, USAGE = P6, ACTUAL=P2  ,$
 FIELDNAME = F2, ALIAS = F2, USAGE = I9, ACTUAL=I4  ,$
 FIELDNAME = F3, ALIAS = F3, USAGE = A2, ACTUAL=A2  ,$
```

The lengths of the ACTUAL attributes for subordinate fields F1, F2, and F3 total 8, which is the length of the ACTUAL attribute of the group key. The lengths of the USAGE attributes for the subordinate fields total 17. However, the length of the group key USAGE attribute is found adding their internal storage lengths as specified by their field types: 8 for USAGE=P6, 4 for USAGE=I9, and 2 for USAGE=A2, for a total of 14.

The GROUP declaration USAGE attribute tells FOCUS how many positions to use to describe the key in a VSAM KSDS file. If a Master File does not completely describe the full key at least once, the following warning message is returned:

```
(FOC1016) INVALID KEY DESCRIPTION IN MASTER FILE
```

The cluster key definition is compared to the Master File for length and displacement.

When you expand on the key in a RECTYPE file, describe the key length in full on the last non-OCCURS segment on each data path.

Do not describe a group with ALIAS=KEY for OCCURS segments. Please see Chapter 5, *Describing Sequential Data Files* for information on describing OCCURS segments.

**Note:** Since all group fields must be defined in alphanumeric format, those which include numeric component fields should not be used as verb objects in a report request.

Going back to our library example, the first field, PUBNO, could actually have been described as a group key. The publisher's number consists of three elements, a number that identifies the publisher, one that identifies the author, and one that identifies the title. Up until now they have been described as a single, ten-digit number made up of these elements. They could have been described as a group key, consisting of a separate field for each element if the file were a VSAM data structure. The FOCUS Master File would look as follows:

```
FILE = LIBRARY5, SUFFIX = VSAM,$
SEGMENT = ROOT, SEGTYPE = S0,$
 GROUP = BOOKKEY     , ALIAS = KEY, USAGE = A10   , ACTUAL = A10  ,$
 FIELDNAME = PUBNO   , ALIAS = PN , USAGE = A3    , ACTUAL = A3   ,$
 FIELDNAME = AUTHNO  , ALIAS = AN , USAGE = A3    , ACTUAL = A3   ,$
 FIELDNAME = TITLNO  , ALIAS = TN , USAGE = A4    , ACTUAL = A4   ,$
 FIELDNAME = AUTHOR  , ALIAS = AT , USAGE = A25   , ACTUAL = A25  ,$
 FIELDNAME = TITLE   , ALIAS = TL , USAGE = A50   , ACTUAL = A50  ,$
 FIELDNAME = BINDING , ALIAS = BI , USAGE = A1    , ACTUAL = A1   ,$
 FIELDNAME = PRICE   , ALIAS = PR , USAGE = D8.2N , ACTUAL = D8   ,$
 FIELDNAME = SERIAL  , ALIAS = SN , USAGE = A15   , ACTUAL = A15  ,$
 FIELDNAME = SYNOPSIS, ALIAS = SY , USAGE = A150  , ACTUAL = A150 ,$
 FIELDNAME = RECTYPE , ALIAS = B  , USAGE = A1    , ACTUAL = A1   ,$
```

When you use a group field with multiple formats in a query, you must account for each position in the group, including trailing blanks or leading zeros. The following example illustrates how to access a group field with multiple formats in a query.

```
GROUP = GRPB, ALIAS = KEY, USAGE = A8, ACTUAL = A8 ,$
 FIELDNAME = FIELD1, ALIAS = F1, USAGE = A2, ACTUAL = A2 ,$
 FIELDNAME = FIELD2, ALIAS = F2, USAGE = I8, ACTUAL = I4 ,$
 FIELDNAME = FIELD3, ALIAS = F3, USAGE = A2, ACTUAL = A2 ,$
```

The values in fields F1 and F3 may include some trailing blanks, and the values in field F2 may include some leading zeros. When using the group in a query, you must account for each position:

```
IF GRPB EQ 'A 0334BB'
```

This introduces the possibility of error. You can eliminate this possibility by using a slash (/) to separate the components of the group key:

```
IF GRPB EQ 'A/334/BB'
```

FOCUS assumes blanks and leading zeros where needed to fill out the key.

# Complex ISAM and Key-Sequenced VSAM Files

Like complex sequential files, complex ISAM and VSAM files can consist of positionally or non-positionally related records. This section will discuss describing both kinds of files to FOCUS, beginning with files with positionally related records.

## Describing Positionally Related Records

Some ISAM and VSAM files are structured so that descendant records relate to each other through concatenating key fields. That is, the key field(s) of a parent record serve as the first part of the key of a child record. In such cases, the segment's key fields must be described to FOCUS using a GROUP declaration. Each segment's GROUP key fields will consist of the renamed key fields from the parent segment plus the unique key field from the child record.

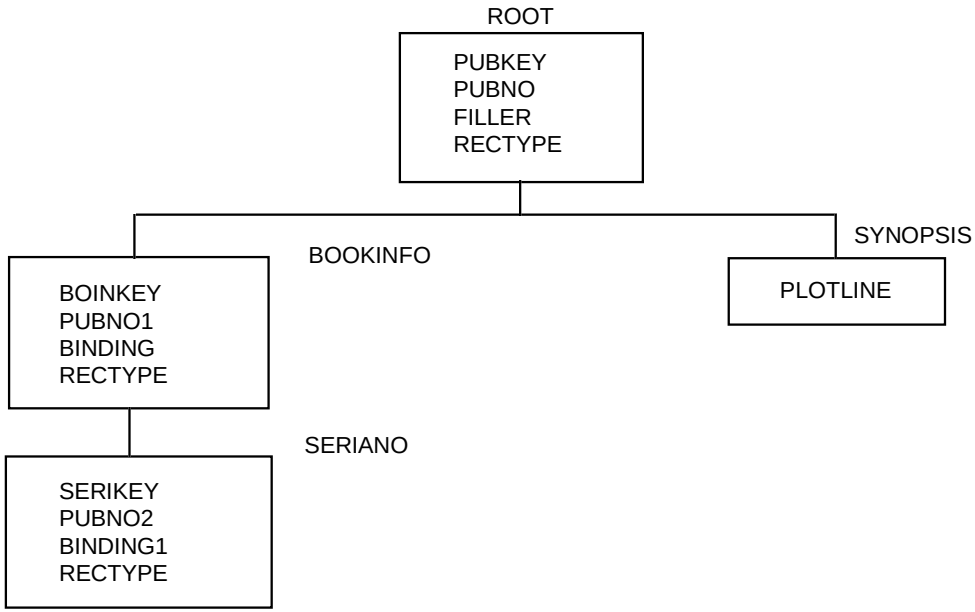
Consider the following VSAM file that contains three types of records. The ROOT records have a key that consists of the publisher's number, PUBNO. The BOOKINFO segment has a key that consists of that same publisher's number, plus a hard or soft-cover indicator, BINDING. The SERIANO segment key consists of the first two elements, plus a record type field, RECTYPE.

The Master File for this structure follows:

```
FILENAME = LIBRARY6, SUFFIX = VSAM,$
SEGNAME = ROOT, SEGTYPE = S0,$
 GROUP=PUBKEY         ,ALIAS=KEY    ,USAGE=A10    ,ACTUAL=A10  ,$
 FIELDNAME=PUBNO      ,ALIAS=PN     ,USAGE=A10    ,ACTUAL=A10  ,$
 FIELDNAME=FILLER     ,ALIAS=       ,USAGE=A1     ,ACTUAL=A1   ,$
 FIELDNAME=RECTYPE    ,ALIAS=1      ,USAGE=A1     ,ACTUAL=A1   ,$
 FIELDNAME=AUTHOR     ,ALIAS=AT     ,USAGE=A25    ,ACTUAL=A25  ,$
 FIELDNAME=TITLE      ,ALIAS=TL     ,USAGE=A50    ,ACTUAL=A50  ,$
SEGNAME=BOOKINFO, PARENT=ROOT, SEGTYPE=S0,$
 GROUP=BOINKEY        ,ALIAS=KEY    ,USAGE=A11    ,ACTUAL=A11  ,$
 FIELDNAME=PUBNO1     ,ALIAS=P1     ,USAGE=A10    ,ACTUAL=A10  ,$
 FIELDNAME=BINDING    ,ALIAS=BI     ,USAGE=A1     ,ACTUAL=A1   ,$
 FIELDNAME=RECTYPE    ,ALIAS=2      ,USAGE=A1     ,ACTUAL=A1   ,$
 FIELDNAME=PRICE      ,ALIAS=PR     ,USAGE=D8.2N  ,ACTUAL=D8   ,$
SEGNAME=SERIANO, PARENT=BOOKINFO, SEGTYPE=S0,$
 GROUP=SERIKEY        ,ALIAS=KEY    ,USAGE=A12    ,ACTUAL=A12  ,$
 FIELDNAME=PUBNO2     ,ALIAS=P2     ,USAGE=A10    ,ACTUAL=A10  ,$
 FIELDNAME=BINDING1   ,ALIAS=B1     ,USAGE=A1     ,ACTUAL=A1   ,$
 FIELDNAME=RECTYPE    ,ALIAS=3      ,USAGE=A1     ,ACTUAL=A1   ,$
 FIELDNAME=SERIAL     ,ALIAS=SN     ,USAGE=A15    ,ACTUAL=A15  ,$
SEGNAME=SYNOPSIS, PARENT=ROOT, SEGTYPE=S0, OCCURS=VARIABLE,$
 FIELDNAME=PLOTLINE   ,ALIAS=PLOTL  ,USAGE=A10    ,ACTUAL=A10  ,$
```

Notice that the length of the key fields specified in the USAGE and ACTUAL attributes of a GROUP declaration is the length of the key fields from the parent segment(s) plus the length of the added field of the child segment (RECTYPE field). In the example above, the length of the GROUP key SERIKEY equals the length of PUBNO2 and BINDING1, the group key from the parent segment, plus the length of RECTYPE, the field added to the group key in the child segment.

In the sample file, the repetition of the publisher's number as PUBNO1 and PUBNO2 in the descendant segments interrelates the three types of records. The file can be diagrammed as the following FOCUS structure:




A typical query might request information on price and call numbers for a specific publisher's number:

```
PRINT PRICE AND SERIAL BY PUBNO
IF PUBNO EQ 1234567890 OR 9876054321
```

Since PUBNO is part of the key, the retrieval can be made quickly and the processing continues. To further speed retrieval you could add search criteria based on the BINDING field, which is also part of the key.

### The RECTYPE Attribute

Complex ISAM and key-sequenced VSAM files also use the RECTYPE attribute to distinguish various record types within the file. Use of the RECTYPE attribute is described in Chapter 5, *Describing Sequential Data Files.*

A parent does not always share its RECTYPE with its descendants. It shares some other identifying piece of information, such as the PUBNO in our example. This is the field that should be included in the parent key, as well as all of its descendants' keys, to relate them.

When using the RECTYPE attribute in ISAM or VSAM files with group keys, the RECTYPE field can only be part of the segment's group key when it belongs to a segment that has no descendants, or to a segment whose descendants are described with an OCCURS attribute. In the previous example, the RECTYPE field is added to the group key in the SERIANO segment, the lowest descendant segment in the chain.

If you place the RECTYPE field in the parent segment's portion of the key, the file will be sorted by record type first. All records with that RECTYPE will be grouped together when FOCUS processes the file, destroying the VSAM positional relationship with records in the child segments. In the previous example, if the RECTYPE of the BOOKINFO segment were made part of the group key, all BOOKINFO records would be sorted together and their relationship to the records in SERIANO would be lost.

## Describing Files With Unrelated Records

Some ISAM and VSAM files do not have records that are related to one another. That is, the ISAM or VSAM key of one record type is independent of the keys of other record types.

For example, consider another VSAM file containing information on our library. This file has three types of records: book information, magazine information, and newspaper information.

There are two possible structures:

- The RECTYPE is the beginning of the key. The key structure is:

| | |
|---|---|
| RECTYPE B | Book Code |
| RECTYPE M | Magazine Code |
| RECTYPE N | Newspaper Code |

  The sequence of records is:

| |
|---|
| Book |
| Book |
| Magazine |
| Magazine |
| Newspaper |
| Newspaper |

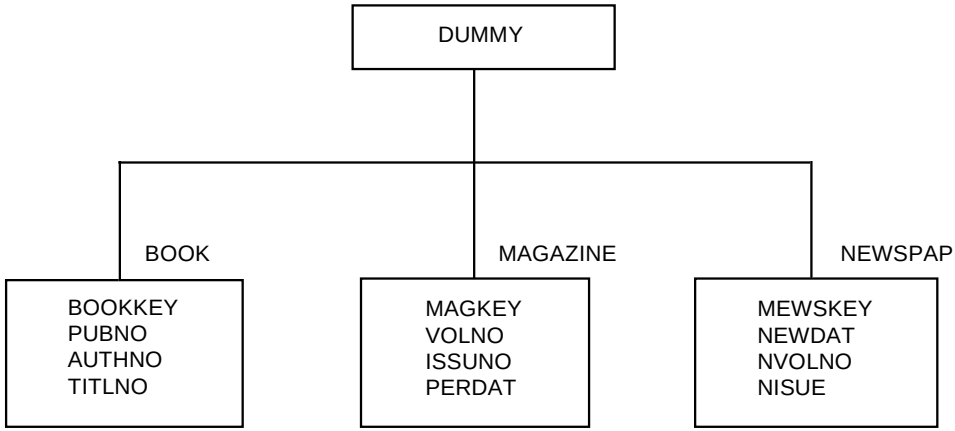
  Note the difference between the use of the RECTYPE here and its use in the previous section. In this case, you do want your records sorted by type first (all the book information together, all the magazine information together, and all the newspaper information together), so it is appropriate to have the RECTYPE as part of the key.

- The RECTYPE is not in the beginning of the key or is outside of the key. The key structure is:

| |
|---|
| Book Code |
| Magazine Code |
| Newspaper Code |

The sequence of record types in the file can be arbitrary.

Both types of file structure can be represented by the following structure:




## How to Describe a Key and Record Type for a File With Unrelated Records

To describe either type of file with unrelated records, make the record types descendants of a dummy segment. The following rules apply to the dummy root segment:

- The segment name must be DUMMY.
- It must have only one field with an empty name and alias.
- It should not have any keys or record types described.
- The USAGE and ACTUAL formats must be A1.

All of the other non-OCCURS segments must point to the dummy root as their parent. Except for the dummy segment, all non-OCCURS segments must describe the full VSAM/ISAM key. If the file does not have a key, the group should not be described. RECTYPEs may be anywhere in the record.

The following is a Master File for the second, more complicated, kind of file with unrelated records:

```
FILE=LIBRARY7, SUFFIX=VSAM,$
SEGMENT=DUMMY,$
 FIELDNAME=        ,ALIAS=      ,USAGE=A1    ,ACTUAL=A1   ,$
SEGMENT=BOOK,PARENT=DUMMY,SEGTYPE=S0,$
GROUP=BOOKKEY      ,ALIAS=KEY   ,USAGE=A11   ,ACTUAL=A11  ,$
 FIELDNAME=PUBNO   ,ALIAS=PN    ,USAGE=A3    ,ACTUAL=A3   ,$
 FIELDNAME=AUTHNO  ,ALIAS=AN    ,USAGE=A3    ,ACTUAL=A3   ,$
 FIELDNAME=TITLNO  ,ALIAS=TN    ,USAGE=A4    ,ACTUAL=A4   ,$
 FIELDNAME=RECTYPE ,ALIAS=B     ,USAGE=A1    ,ACTUAL=A1   ,$
 FIELDNAME=AUTHOR  ,ALIAS=AT    ,USAGE=A25   ,ACTUAL=A25  ,$
 FIELDNAME=TITLE   ,ALIAS=TL    ,USAGE=A50   ,ACTUAL=A50  ,$
 FIELDNAME=BINDING ,ALIAS=BI    ,USAGE=A1    ,ACTUAL=A1   ,$
 FIELDNAME=PRICE   ,ALIAS=PR    ,USAGE=D8.2N ,ACTUAL=D8   ,$
 FIELDNAME=SERIAL  ,ALIAS=SN    ,USAGE=A15   ,ACTUAL=A15  ,$
 FIELDNAME=SYNOPSIS,ALIAS=SY    ,USAGE=A150  ,ACTUAL=A150 ,$
SEGMENT=MAGAZINE, PARENT=DUMMY, SEGTYPE=S0,$
GROUP=MAGKEY       ,ALIAS=KEY   ,USAGE=A11   ,ACTUAL=A11  ,$
 FIELDNAME=VOLNO   ,ALIAS=VN    ,USAGE=A2    ,ACTUAL=A2   ,$
 FIELDNAME=ISSUNO  ,ALIAS=IN    ,USAGE=A2    ,ACTUAL=A2   ,$
 FIELDNAME=PERDAT  ,ALIAS=DT    ,USAGE=A6    ,ACTUAL=A6   ,$
 FIELDNAME=RECTYPE ,ALIAS=M     ,USAGE=A1    ,ACTUAL=A1   ,$
 FIELDNAME=PER_NAME,ALIAS=PRN   ,USAGE=A50   ,ACTUAL=A50  ,$
SEGMENT=NEWSPAP, PARENT=DUMMY, SEGTYPE=S0,$
GROUP=NEWSKEY      ,ALIAS=KEY   ,USAGE=A11   ,ACTUAL=A11  ,$
 FIELDNAME=NEWDAT  ,ALIAS=ND    ,USAGE=A6    ,ACTUAL=A6   ,$
 FIELDNAME=NVOLNO  ,ALIAS=NV    ,USAGE=A2    ,ACTUAL=A2   ,$
 FIELDNAME=NISSUE  ,ALIAS=NI    ,USAGE=A2    ,ACTUAL=A2   ,$
 FIELDNAME=RECTYPE ,ALIAS=N     ,USAGE=A1    ,ACTUAL=A1   ,$
 FIELDNAME=NEWNAME ,ALIAS=NN    ,USAGE=A50   ,ACTUAL=A50  ,$
```

## VSAM Repeating Groups With RECTYPEs

If a file contains records that have repeating groups, the OCCURS attribute is used to describe a separate segment for the repeating fields.

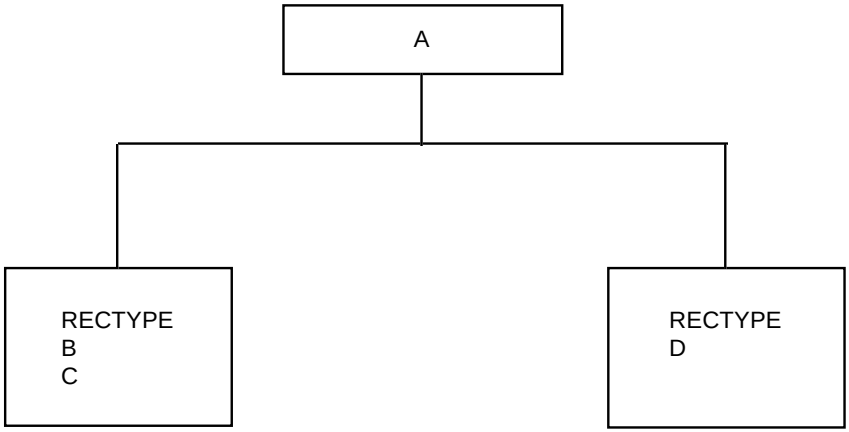
**Note:** OCCURS segments are treated as MISSING if they are not activated when INCLUDEd in a FOCUS MODIFY. A period (.) is placed in the first byte of the first occurrence of OCCURS=n or OCCURS=field name to signify that the segment is missing.

In some files, however, the repeating fields themselves must be identified according to a RECTYPE indicator.

Suppose you want to describe a file that, schematically, looks like this:

| | | | | | |
|---|---|---|---|---|---|
| A | RECTYPE | B C | RECTYPE | B C | |
| A | RECTYPE | D | RECTYPE | D | |

You need to describe three segments in your Master File, with A as the root segment, and segments for B, C, and D as two descendant OCCURS segments for A:




Each of the two descendant OCCURS segments in this example depends on the RECTYPE indicator that appears for each occurrence.

All the rules of syntax for using RECTYPE fields and OCCURS segments also apply to RECTYPEs within OCCURS segments.

Since the OCCURS segments depend on the RECTYPE indicator for its evaluation, the RECTYPE must appear at the start of each OCCURS segment. This allows very complex files to be described, including those with nested and parallel repeating groups that depend on RECTYPEs.

In the next example, B/C, and D represent a nested repeating group, and E represents a parallel repeating group.

| | | | | |
|---|---|---|---|---|
| A | RECTYPE B C | RECTYPE D | RECTYPE E | RECTYPE E |

The Master File would be coded as:

```
FILENAME=SAMPLE,SUFFIX=VSAM,$
SEGNAME=ROOT,SEGTYPE=S0,$
 GROUP=GRPKEY      ,ALIAS=KEY   ,USAGE=A8    ,ACTUAL=A8         ,$
 FIELD=FLD000      ,E00         ,A08         ,A08               ,$
 FIELD=A_DATA      ,E01         ,A02         ,A02               ,$
SEGNAME=SEG001,PARENT=ROOT,OCCURS=VARIABLE,SEGTYPE=S0           ,$
 FIELD=RECTYPE     ,A01         ,A01         ,ACCEPT=B OR C     ,$
 FIELD=B_OR_C_DATA ,E02         ,A08         ,A08               ,$
SEGNAME=SEG002,PARENT=SEG001,OCCURS=VARIABLE,SEGTYPE=S0         ,$
 FIELD=RECTYPE     ,D           ,A01         ,A01               ,$
 FIELD=D_DATA      ,E03         ,A07         ,A07               ,$
SEGNAME=SEG003,PARENT=ROOT,OCCURS=VARIABLE,SEGTYPE=S0           ,$
 FIELD=RECTYPE     ,E           ,A01         ,A01               ,$
 FIELD=E_DATA      ,E04         ,A06         ,A06               ,$
```

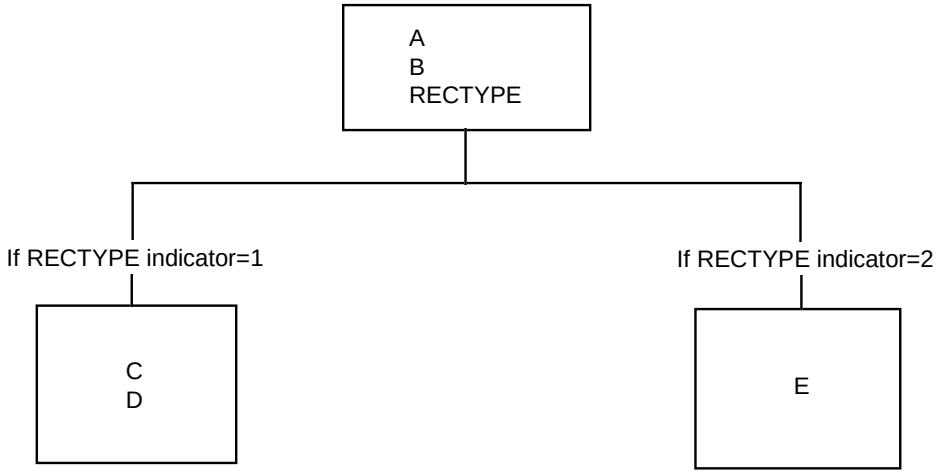
# Describing VSAM Repeating Groups Using MAPFIELD

In another possible combination of RECTYPE and OCCURS, a record contains a RECTYPE indicator that is followed by a repeating group. Schematically, the record would appear like this:

| | | | | |
|---|---|---|---|---|
| A B | RECTYPE (1) | C D | C D | C D |
| A B | RECTYPE (2) | E E | | |

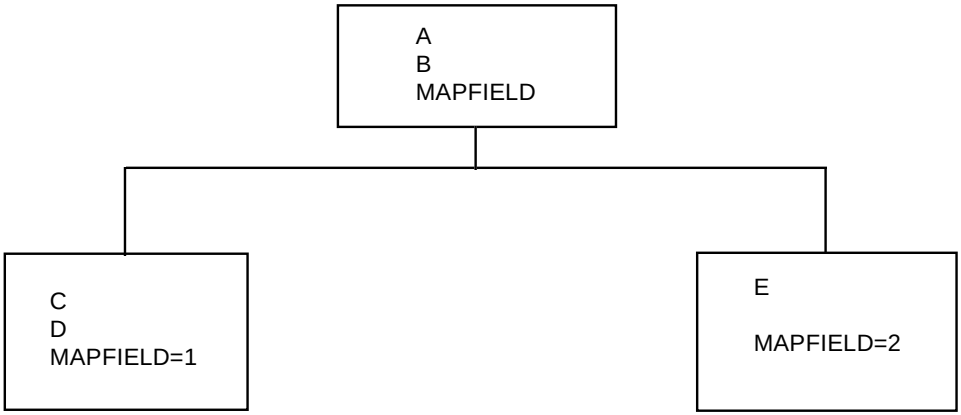
The first record contains “header” information, values for A and B, followed by an OCCURS segment of C and D that was identified by its preceding RECTYPE indicator. The second record has a different RECTYPE indicator and contains a different repeating group, this time for E.

The following diagram illustrates this relationship.




Since the OCCURS segments are identified by the RECTYPE indicator rather than the parent A/B segment, you can use the keyword MAPFIELD. MAPFIELD identifies a field in the same way RECTYPE does, but since the OCCURS segments will each have their own values for MAPFIELD, the value of MAPFIELD is associated with each OCCURS segment by means of a complementary field named MAPVALUE.

The following diagram illustrates this relationship:




MAPFIELD is assigned as the ALIAS of the field that will be the RECTYPE indicator. You can give this field any name. It is otherwise described according to the usual syntax

```
FIELD = name, ALIAS = MAPFIELD, USAGE = format, ACTUAL = format,$
```

where:

*name*
: The name you choose to provide for this field.

`ALIAS`
: MAPFIELD is assigned as the alias of the field that will be the RECTYPE indicator.

`USAGE`
: Follows the usual field format.

`ACTUAL`
: Follows the usual field format.

The descendant segment values depend on the value of the MAPFIELD. They are described as separate segments, one for each possible value of MAPFIELD, and all descending from the segment that has the MAPFIELD. A special field, MAPVALUE, is described as the last field in these descendant segments after the ORDER field, if one has been used. The actual MAPFIELD value is supplied as the ALIAS of the MAPVALUE.

The syntax is

```
FIELD = MAPVALUE, ALIAS = alias, USAGE = format, ACTUAL = format,
ACCEPT = {list|range} ,$
```

where:

`MAPVALUE`
: Indicates that the segment depends on a MAPFIELD in its parent segment.

*`alias`*
: Is the primary MAPFIELD identifier. If there is an ACCEPT list, this value is any value in the ACCEPT list or range.

`USAGE`
: Is the same format as the MAPFIELD format in the parent segment.

`ACTUAL`
: Is the same format as the MAPFIELD format in the parent segment.

*`list`*
: Is the list of one or more lines of specified MAPFIELD values for records that have the same segment layout. The maximum number of characters allowed in the list is 255. Each item in the list must be separated by either a blank or the keyword OR. If the list contains embedded blanks or commas, it must be enclosed within single quotation marks(''). The list may contain a single MAPFIELD value.

  For example:

  ```
  FIELDNAME = MAPFIELD, ALIAS = A, USAGE = A1, ACTUAL = A1,
  ACCEPT = A OR B OR C,$
  ```

*`range`*
: Is a range of one or more lines of MAPFIELD values for records that have the same segment layout. The maximum number of characters allowed in the range is 255. If the range contains embedded blanks or commas, it must be enclosed in single quotation marks (').

  To specify a range of values, include the lowest value, the keyword TO, and the highest value, in that order.

Using the sample file at the beginning of this section, the Master File for this file looks like this:

```
FILENAME=EXAMPLE,SUFFIX=FIX,$
 SEGNAME=ROOT,SEGTYPE=S0,$
 FIELD =A  ,                    ,A14  ,A14              ,$
 FIELD =B  ,                    ,A10  ,A10              ,$
 FIELD =FLAG   ,MAPFIELD        ,A01  ,A01              ,$
 SEGNAME=SEG001,PARENT=ROOT,OCCURS=VARIABLE,SEGTYPE=S0   ,$
 FIELD =C  ,                    ,A05  ,A05              ,$
 FIELD =D  ,                    ,A07  ,A07              ,$
 FIELD =MAPVALUE ,1             ,A01  ,A01              ,$
 SEGNAME=SEG002,PARENT=ROOT,OCCURS=VARIABLE,SEGTYPE=S0   ,$
 FIELD =E  ,                    ,D12.2,D8               ,$
 FIELD =MAPVALUE ,2             ,A01  ,A01              ,$
```

**Note:** MAPFIELD can only exist on an OCCURS segment that has not been re-mapped. This means that the segment definition cannot contain POSITION=fieldname.

As shown in the example above, MAPFIELD and MAPVALUE may be used with suffix=FIX and suffix=VSAM files.

## VSAM Data and Index Buffers

Two SET commands make it possible to establish DATA and INDEX buffers for processing VSAM files online.

The AMP sub-parameters BUFND and BUFNI allow MVS BATCH users to enhance the I/O efficiency of TABLE, TABLEF, MODIFY, and JOIN against VSAM files by holding frequently used VSAM Control Intervals in memory, rather than on physical DASD. By reducing the number of physical Input/Output operations, job throughput is improved. The new SET commands allow FOCUS users (in CMS, MVS/TSO, and MSO) to realize similar performance gains in interactive sessions. In general, BUFND (data buffers) increase the efficiency of physical sequential reads, whereas BUFNI (index buffers) are most beneficial in JOIN or KEYED access operations.

The syntax is

```
{MVS|CMS} VSAM SET BUFND {n|8}
{MVS|CMS} VSAM SET BUFNI {n|1}
```

where:

*n*

Is the number of data or index buffers. The default values are BUFND=8, BUFNI=1 (eight data buffers and one index buffer).

To determine how many buffers are in effect at any time, issue the query:

```
{MVS|CMS} VSAM SET ?
```

# Alternate Indexes

FOCUS supports the use of alternate indexes (keys) with VSAM key-sequenced files. A key-sequenced VSAM file consists of two elements: an index component and a data component. The data component contains the actual data records, while the index component is the key used to locate the data records in the file. Together, these two components are referred to as the "base cluster."

An alternate index is a separate, additional index structure that allows you to access records in a KSDS VSAM file based on a key other than the file's primary key. For instance, you may usually use a personnel file sequenced by Social Security number, but have an occasional need to have the records retrieved sorted by job description. The job description field might be described as an alternate index. An alternate index must be related to the base cluster it describes by a "path," which is stored in a separate file.

The alternate index is a VSAM structure and is, consequently, created and maintained in the VSAM environment. It can, however, be described to FOCUS in your Master File, so that you can take advantage of the benefits of an alternate index in the FOCUS environment.

The primary benefit of these indexes is improved efficiency. You can use it as an alternate, more efficient, retrieval sequence or you can take advantage of its potential indirectly, with screening tests (IF…LT, IF…LE, IF…GT, IF…GE, IF…EQ, IF…FROM…TO, IF…IS) which are translated into direct reads against the alternate index. You can also join files with the JOIN command through this alternate index.

It is not necessary to explicitly identify the indexed view in order to take advantage of the alternate index. FOCUS automatically selects an alternate index when one is described to FOCUS in the Master File.

To take advantage of a specific alternate index during a TABLE request, provide an IF or WHERE test on the alternative index field that meets the above criteria. For example:

```
TABLE FILE CUST
PRINT SSN
WHERE LNAME EQ 'SMITH'
END
```

As you will see in the Master File in *Describing Alternate Indexes* on page 6-17, the LNAME field is defined as an alternate index field. The records in the file will be retrieved according to their last names, and certain IF screens on the field LNAME will result in direct reads. Note that if the alternate index field name is omitted, the primary key (if there is any) will be used for a sequential or a direct read, and the alternate indexes will be treated as regular fields.

## Describing Alternate Indexes

Alternate indexes must be described in the Master File as fields with FIELDTYPE=I. The ALIAS of the alternate index field must be the file name allocated to the corresponding path name. Alternate indexes can be described as GROUPs if they consist of portions with dissimilar formats. Remember that the ALIAS=KEY must be used to describe the primary key.

Consider the following example:

```
FILENAME = CUST, SUFFIX = VSAM,$
SEGNAME = ROOT, SEGTYPE = S0,$
 GROUP = G, ALIAS = KEY, A10, A10,$
 FIELD = SSN,   SSN, A10, A10,$
 FIELD = FNAME, DD1, A10, A10, FIELDTYPE=I,$
 FIELD = LNAME, DD2, A10, A10, FIELDTYPE=I,$
```

In this example, SSN is a primary key and FNAME and LNAME are alternate indices. The path data set must be allocated to the ddname specified in ALIAS= of your alternate index field. In this Master File, ALIAS=DD1 and ALIAS=DD2 would each have an allocation pointing to the path data set. FNAME and LNAME must have INDEX=I or FIELDTYPE=I coded in the Master File. CUST must be allocated to the base cluster.

Only one record type can be referred to in the request when alternate indexes are used, but the number of OCCURS segments is unrestricted.

Note that the path name in the allocation will be different from both the cluster name and the alternate index name.

If you are not sure of the path names and alternate indexes associated with a given base cluster, you can use the IDCAMS utility. (See the IBM manual entitled *Using VSAM Commands and Macros* for details.)

The following example demonstrates how to find the alternate index and path names associated with a base cluster named CUST.DATA:

First, find the alternate index names (AIX) associated with the given cluster.

```
IDCAMS input:
 LISTCAT CLUSTER ENTRIES(CUST.DATA) ALL

IDCAMS output (fragments):
 CLUSTER -------- CUST.DATA
  ASSOCIATIONS
   AIX ---------- CUST.INDEX1
   AIX ---------- CUST.INDEX2
```

This gives you the names of the alternate indexes (AIX): CUST.INDEX1 and CUST.INDEX2.

Next, find the path names associated with the given AIX name.

```
IDCAMS input:
 LISTCAT AIX ENTRIES (CUST.INDEX1 CUST.INDEX2) ALL

IDCAMS output (fragments):
 AIX ---------CUST.INDEX1
  ASSOCIATIONS
   CLUSTER -- CUST.DATA
   PATH ------CUST.PATH1
 AIX ---------CUST.INDEX2
  ASSOCIATIONS
   CLUSTER -- CUST.DATA
   PATH ------CUST.PATH2
```

This gives you the path names: CUST.PATH1 and CUST.PATH2.

This information along with the TSO DDNAME command may be used to ensure the proper allocation of your alternate index.

# 7 Describing FOCUS Databases

**Topics:**

- Designing FOCUS Databases
- Describing Single Segments
- Describing Individual Fields

FOCUS is a universal application development language and decision support system that enables you to use many different types of data sources. FOCUS also provides its own type of data source, the FOCUS database, which is well suited for sophisticated application development.

The following topics cover data description topics unique to FOCUS databases:

- **Design tips.** *Designing FOCUS Databases* on page 7-2, provides some suggestions for people who are designing a new FOCUS database or changing the design of an existing database.
- **Describing segments.** *Describing Single Segments* on page 7-5, contains information about Master File segment declarations for FOCUS databases, including defining segment relationships, keys, and sort order using the SEGTYPE attribute, and storing segments in different locations using the LOCATION attribute. Chapter 8, *Defining a Join in a Master File*, explains how to define static and dynamic joins in a Master File.
- **Describing fields.** *Describing Individual Fields* on page 7-13 contains information about Master File field declarations for FOCUS databases, including the FIND option of the ACCEPT attribute; indexing fields using the INDEX attribute; and the internal storage requirements of each data type defined by the FORMAT attribute, and of null values described by the MISSING attribute.

# Designing FOCUS Databases

The FOCUS database management system enables you to create sophisticated hierarchical data structures. The following sections provide information to help you design an effective and efficient FOCUS database and tell you how you can change the design after the database has been created.

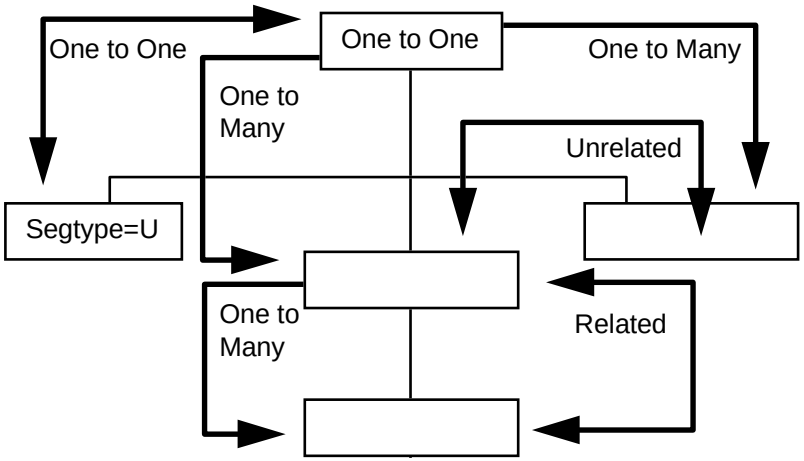
## Data Relationships

The primary consideration when designing a database is the set of relationships among the various fields. Before you create the Master File, you may wish to draw a diagram of these relationships. Is a field related to any other fields? If so, is it a one-to-one or a one-to-many relationship? If any of the data already exists in another data file, can that data file be joined to this one?

In general, you can use the following guidelines:

- All information that occurs once for a given record should be placed in the root segment or a unique child segment.
- Any information that can be retrieved from a joined data file should, in most cases, be retrieved in this way, and not redundantly maintained in two different files.
- Any information that has a many-to-one relationship with the information in a given segment should be stored in a descendant of that segment.
- Related data in child segments should be stored in the same path; unrelated data should be placed in different paths.

The following illustration summarizes the rules for data relationship considerations:



## Join Considerations

If you plan to join one segment to another, remember that both the host and cross-referenced fields must have the same format, and the cross-referenced field must be indexed using the INDEX attribute. In addition, for a cross-reference in a Master File, the host and cross-referenced fields must share the same name—that is, the name of both fields, or the alias of both fields must be identical, or else the name of one field must be identical to the alias of the other.

## General Efficiency Considerations

FOCUS processes a FOCUS database by reading the root segment first then traversing the hierarchy to satisfy you query. The smaller you make the root segment, the more root segment instances FOCUS can read at one time, and the faster FOCUS can select records and process a query.

You can also improve record substitution efficiency by setting AUTOPATH. AUTOPATH is the automation of TABLE FILE *ddname.fieldname* syntax, where field name is not indexed and physical retrieval starts at the field name segment. AUTOPATH is described in the *Developing Applications* manual.

As with most information processing issues, there is a trade-off when designing an efficient FOCUS database: you need to balance the desire to speed up record *retrieval*, by reducing the size of the root segment, against the need to speed up record selection, by placing fields used in record selection tests as high in the data structure as possible. The placement of selection fields is important because when a field fails a record selection test—that is, a WHERE or IF test—in a report request, FOCUS uses time efficiently by avoiding any additional work with that segment instance and ignoring all of its descendant instances. The higher the selection fields are in a data structure, the fewer the number of segments that need to be read to determine a record's status, and the greater the number of segments that will be ignored—consuming no processing time—when a record fails a selection test.

After you have designed and created a database, if you want to select records based on fields that are low in the data structure, you can rotate the data structure to temporarily place those fields higher by using an alternate view. Alternate views are discussed in *Rotating a Database: Alternate Views* in Chapter 3, *Describing Groups of Fields*. Using alternate views in report requests is discussed in the *Creating Reports* manual.

The following guidelines will help you to design an efficient data structure:

- Limit the information in the root segment to what is necessary to identify the record and to what is used often in screening conditions.
- Avoid unnecessary key fields. Segments with a SEGTYPE of S1 will be processed much more efficiently than those with, for example, a SEGTYPE of S9.
- Index the first field of the segment (the key field) if the root segment of your file is SEGTYPE S1 for increased efficiency in MODIFY procedures that read transactions from unsorted files (FIXFORM).
- Use segments with a SEGTYPE of SH1 when adding and maintaining data in date sequence. A SEGTYPE of SH1 puts each new record at the beginning of the file, not at the end.
- If a segment contains fields frequently used in record selection tests, keep the segment small by limiting it to key fields, selection fields, and other fields frequently used in reports.
- Index fields on which you perform frequent searches of unique instances. When you specify that a field is indexed, FOCUS constructs and maintains a table of data values and their corresponding physical locations in the file. Thus, indexing a field speeds retrieval.

  You can index any field you want, although it is advisable to limit the number of indices in a database since each index requires additional storage space. You will need to weigh the increase in speed against the increase in space.

## Changing an Existing Database

Once you have designed and created a FOCUS database, you can change some of its characteristics simply by editing the corresponding attribute in the Master File. The documentation for each attribute specifies whether it can be edited after the database has been created.

Some characteristics whose attributes cannot be edited *can* be changed if you rebuild the database using the REBUILD facility, as described in the *Maintaining Databases* manual. You can also use REBUILD to add new fields to a database.

# Describing Single Segments

LOCATION segments may be coded in a Master File to expand the file size by pointing to another physical file location. LOCATION is discussed in Chapter 3, *Describing Groups of Fields.*

Three additional segment attributes that describe joins between FOCUS segments, CRFILE, CRKEY, and CRSEGNAME, are described in Chapter 8, *Defining a Join in a Master File*.

## Maximum Number of Segments

The number of segments cannot exceed 64.

## Describing Keys, Sort Order, and Segment Relationships: SEGTYPE

FOCUS databases use the SEGTYPE attribute to describe a segment's key fields and sort order, as well as the relationship of the segment to its parent.

The SEGTYPE attribute is also used with suffix=FIX files to indicate a logical key sequence for that file. SEGTYPE is discussed in Chapter 3, *Describing Groups of Fields.*

### *Syntax* How to Describe a Segment

The syntax of the SEGTYPE attribute when used for a FOCUS database is:

```
SEGTYPE = segtype
```

Valid values are:

`SH[n]`
: Indicates that the segment's instances are sorted from the highest value to the lowest value, based on the value of the first n fields in the segment. *n* can be any number from 1 to 99; if you do not specify it, it defaults to 1.

`S[n]`
: Indicates that the segment's instances are sorted from the lowest value to the highest value, based on the value of the first n fields in the segment. *n* can be any number from 1 to 255; if you do not specify it, it defaults to 1.

`S0`
: Indicates that the segment has no key fields and is therefore not sorted by FOCUS. New instances are added to the end of the segment chain.

  S0 segments are often used to store text for applications where the text will need to be retrieved in the order entered and the application does not need to search for particular instances.

`b̸ (blank)`

A SEGTYPE value of blank indicates that the segment has no key fields and is therefore not sorted. New instances are added to the end of the segment chain.

SEGTYPE = b̸ segments are often used in situations where there are very few segment instances and the information stored in the segment does not include a field that can serve as a key.

Note that a b̸ segment cannot be a root segment.

`U`

Indicates that the segment is unique—that is, it has a one-to-one relationship to its parent. Note that a unique segment described with a SEGTYPE of U cannot have any children.

`KM`

Indicates that this is a cross-referenced segment joined to the database using a static join defined in the Master File and has a one-to-many relationship to the host segment. Joins defined in the Master File are described in Chapter 8, *Defining a Join in a Master File.*

`KU`

Indicates that this is a cross-referenced segment joined to the database using a static join defined in the Master File and has a one-to-one relationship to the host segment (that is, it is a unique segment). Joins defined in the Master File are described in Chapter 8, *Defining a Join in a Master File.*

`DKM`

Indicates that this is a cross-referenced segment joined to the database using a dynamic join defined in the Master File and has a one-to-many relationship to the host segment. Joins defined in the Master File are described in Chapter 8, *Defining a Join in a Master File.*

`DKU`

Indicates that this is a cross-referenced segment joined to the database using a dynamic join defined in the Master File and has a one-to-one relationship to the host segment (that is, it is a unique segment). Joins defined in the Master File are described in Chapter 8, *Defining a Join in a Master File.*

`KL`

Indicates that this segment is described in a Master File defined join as descending from a KM, KU, DKM, or DKU segment in a cross-referenced database and has a one-to-many relationship to its parent.

`KLU`

Indicates that this segment is described in a Master File defined join as descending from a KM, KU, DKM, or DKU segment in a cross-referenced database and has a one-to-one relationship to its parent (that is, it is a unique segment).

*Reference*

## Usage Notes for SEGTYPE

Note the following rules when using the SEGTYPE attribute with a FOCUS database:

- **Alias.** SEGTYPE does not have an alias.
- **Changes.** You can change a SEGTYPE of S[n] or SH[n] to S0 or b. To make any other change to SEGTYPE you need to use the REBUILD facility. REBUILD is described in the *Maintaining Databases* manual.

### Describing Keys

You use the SEGTYPE attribute to describe which fields in a segment are key fields. The values of these fields determine how the segment instances are sequenced. The keys must be the first fields in a segment. You can specify up to 255 keys in a segment that is sorted from low to high (SEGTYPE = Sn), and up to 99 keys in a segment sorted from high to low (SEGTYPE = SHn). To maximize efficiency, it is recommended that you specify only as many keys as you need to make each record unique. You can also choose not to have any keys (SEGTYPE = S0 and SEGTYPE = b (blank).

**Note:** Text fields cannot be used as key fields.

### Describing Sort Order

For segments that have key fields, you use the SEGTYPE attribute to describe the segment's sort order. You can sort a segment's instances in two ways:

- **Low to high.** By specifying a SEGTYPE of Sn (where n is the number of keys), FOCUS sorts the instances using the concatenated values of the first n fields, beginning with the lowest value and continuing to the highest value.
- **High to low.** By specifying a SEGTYPE of SHn (where n is the number of keys), FOCUS sorts the instances using the concatenated values of the first n fields, beginning with the highest value and continuing to the lowest value.

  Segments whose key is a date field often use a high-to-low sort order, since it ensures that the segment instances with the most recent dates will be the first ones encountered in a segment chain.

*Example* **Understanding Sort Order**

For example, suppose the following fields in a segment represent a department code and the employee's last name:

| | | | | |
|---|---|---|---|---|
| 06345 | 19887 | 19887 | 23455 | 21334 |
| Jones | Smith | Frank | Walsh | Brown |

If you set SEGTYPE to S1, the department code becomes the key. (Note that two records have duplicate key values in order to illustrate a point about S2 segments later in this example; duplicate key values are not recommended for S1 and SH1 segments.) The segment instances will be sorted as follows:

| | | | | |
|---|---|---|---|---|
| 06345 | 19887 | 19887 | 21334 | 23455 |
| Jones | Smith | Frank | Brown | Walsh |

If you change the field order to put the last name field before the department code and leave SEGTYPE as S1, the last name becomes the key. The segment instances will be sorted as follows:

| | | | | |
|---|---|---|---|---|
| Brown | Frank | Jones | Smith | Walsh |
| 21334 | 19887 | 06345 | 19887 | 23455 |

Alternately, if you leave the department code as the first field, but set SEGTYPE to S2, the segment will be sorted first by the department code and then by last name, as follows:

| | | | | |
|---|---|---|---|---|
| 06345 | 19887 | 19887 | 21334 | 23455 |
| Jones | Frank | Smith | Brown | Walsh |

## Describing Segment Relationships

The SEGTYPE attribute describes the relationship of a segment to its parent segment:

- Physical one-to-one relationships are usually specified by setting SEGTYPE to U. If a segment is described in a Master File defined join as descending from the cross-referenced segment, then in the join description, SEGTYPE is set to KLU.
- Physical one-to-many relationships are specified by setting SEGTYPE to any valid value beginning with S (such as S0, SHn, and Sn), to blank, or, if a segment is described in a Master File defined join as descending from the cross-referenced segment, to KL.
- One-to-one joins defined in a Master File are specified by setting SEGTYPE to KU or DKU, as described in Chapter 8, *Defining a Join in a Master File.*
- One-to-many joins defined in a Master File are specified by setting SEGTYPE to KM or DKM, as described in Chapter 8, *Defining a Join in a Master File.*

# Storing Segments in Different Locations: LOCATION

By default, all of the segments in a FOCUS database are stored in one physical file. For example, all of the EMPLOYEE database's segments are stored in the file named EMPLOYEE. If you wish, you can use the LOCATION attribute to specify that one or more segments be stored in a physical file separate from the main database file. You can use a total of 64 LOCATION files per Master File (one LOCATION attribute per segment). This can be helpful if you want to create a database larger than the FOCUS limit for a single database file, or if you want to store parts of the database in separate locations for security or other reasons.

There are at least two cases in which you may want to use the LOCATION attribute. First, each physical file is individually subject to a maximum file size of one gigabyte. You can use the LOCATION attribute to increase the size of your database by splitting it into several physical files, each one subject to the one gigabyte maximum. (You may also want to read Chapter 8, *Defining a Join in a Master File*, to see if it would be more efficient to structure your data as several joined databases.)

You can also store your data in separate physical files to take advantage of the fact that only the segments needed for a report must be present. Unreferenced segments stored in separate files can be kept on separate storage media to save space or implement separate security mechanisms. In some situations, the separation of the segments into different files allows different disk drives to be used.

Divided databases do require more careful file maintenance. You have to be especially careful about procedures that are done separately to separate files, such as backups. If you do backups on Tuesday and Thursday for two related files, and you restore the FOCUS structure using the Tuesday backup for one half and the Thursday backup for the other, FOCUS has no way of detecting this discrepancy.

The LOCATION attribute can be changed if the existing external file name (ddname) can be changed.

## *Syntax* How to Store Segments in Different Locations

The syntax of the LOCATION attribute is

```
LOCATION = filename
```

where:

*filename*
Is the ddname of the file in which the segment is to be stored.

For example:

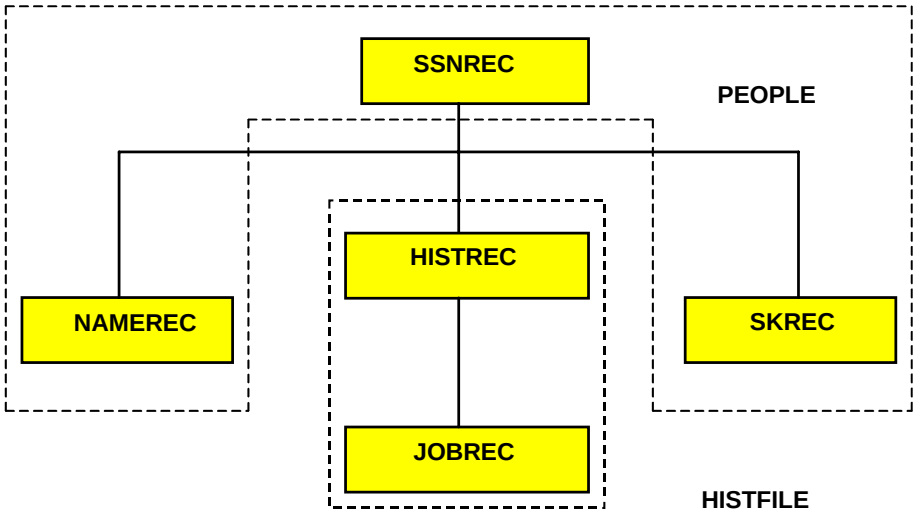
```
SEGNAME = HISTREC, SEGTYPE = S1, PARENT = SSNREC, LOCATION = HISTFILE, $
```

### *Example* Specifying Locations for Segments

The following example illustrates the use of this feature:

```
FILENAME = PEOPLE, SUFFIX = FOC, $

SEGNAME = SSNREC,       SEGTYPE = S1, $
FIELD = SSN,            ALIAS = SOCSEG,   USAGE = I9, $

SEGNAME = NAMEREC,      SEGTYPE = U,      PARENT = SSNREC, $
FIELD = LNAME,          ALIAS = LN,       USAGE = A25, $

SEGNAME = HISTREC,      SEGTYPE = S1,     PARENT = SSNREC,
LOCATION = HISTFILE, $
FIELD = DATE,           ALIAS = DT,       USAGE = YMD, $

SEGNAME = JOBREC,       SEGTYPE = S1,     PARENT = HISTREC, $
FIELD = JOBCODE,        ALIAS = JC,       USAGE = A3, $

SEGNAME = SKREC,        SEGTYPE = S1,     PARENT = SSNREC, $
FIELD = SCODE,          ALIAS = SC,       USAGE = A3, $
```

This description groups the five segments into two physical files, as shown in the following diagram:




Note that the segment named SKREC, which contains no LOCATION attribute, is stored in the PEOPLE file. This occurs because if no LOCATION attribute is specified for a segment, it is placed by default in the same file as its parent. In our example, you could assign the SKREC segment to a different file by specifying the LOCATION attribute in its declaration.

## Separating Large Text Fields

Text fields, by default, are stored in one physical file with non-text fields. However, as with segments, a text field can be located in its own physical file or any combination of text fields can share one or several physical files. You specify that you want a text field stored in a separate file by using the LOCATION attribute in the field definition.

For example:

```
FIELD = DESCRIPTION, ALIAS = CDESC, USAGE = TX50, LOCATION = CRSEDESC ,$
```

The text for DESCRIPTION will be stored in a separate physical file named CRSEDESC.

**Note:** USAGE may equal TXnnF. “F” is used to format the text field for redisplay when TED is called using ON MATCH or ON NOMATCH in MODIFY. For more information, see the *Maintaining Databases* manual.

If you have more than one text field, each field can be stored in its own file, or several text fields can be stored in one file:

```
FIELD = DESCRIPTION , ALIAS = CDESC,  USAGE = TX50, LOCATION = CRSEDESC,$
FIELD = PREREQUISITE, ALIAS = PREEQ,  USAGE = TX50, LOCATION = PREREQS ,$
FIELD = TOPICS,       ALIAS =      ,  USAGE = TX50, LOCATION = CRSEDESC,$
```

In this example, the text fields DESCRIPTION and TOPICS are stored in the LOCATION file CRSEDESC. The text field PREREQUISITE is stored in another file, PREREQS.

As with segments, you might want to use the LOCATION attribute on a text field if it is very long. However, unlike LOCATION segments, LOCATION files for text fields must be present during a request, whether or not the text field is referenced.

The LOCATION attribute can be used independently for segments and for text fields. That is, you can use the LOCATION attribute for a text field without using it for a segment. You can also use the LOCATION attribute for both the segment and the text field in the same Master File.

**Note:** Field names for text fields in FOCUS Master Files are limited to 12 characters; however alias names for these fields can be up to 66 characters.

## Limits on the Number of LOCATION Files

There is a limit on the number of different location segments and text location files you can specify. This limit is based on the number of entries allowed in the file directory table (FDT) for FOCUS files. The FDT contains the names of the segments in the file, the names of indexed fields, and the names of location files for text fields. The FDT can contain 189 entries of which up to 64 can represent segments and location files. Each unique location file counts as one entry in the FDT.

For a given FOCUS file, the maximum number of location files can be determined by the following formula

```
Available FDT entries = 189 - (Segments + Indexes)
```

```
Location files = min (64, Available FDT entries)
```

where:

*Location files*
: Is the maximum number of location segments and text location files (up to a maximum of 64).

*Segments*
: Is the number of segments in the Master File.

*Indexes*
: Is the number of indexed fields.

For example, a ten-segment file with 2 indexed fields would enable you to specify up to 52 location segments and/or location files for text fields (189 - (10 + 2)). Using the formula, the result would equal 177; however, the maximum number of text location files must always be *no more than 64*.

**Note:** If you specify a text field with a LOCATION attribute, the main file will be included in the text location file count.

# Describing Individual Fields

There are two field attributes that have special values or are unique to FOCUS databases: ACCEPT and INDEX (also known as FIELDTYPE). This section describes both of these, and also documents the internal storage formats of each FORMAT data type and the internal values used with the MISSING attribute.

## The ACCEPT Attribute

ACCEPT is an optional attribute that you can use to validate data as it is entered into a field using a MODIFY procedure. Its use with all types of data files is described in Chapter 4, *Describing Individual Fields*. However, ACCEPT has a special option, FIND, that you can only use with FOCUS databases. FIND enables you to verify incoming data against values stored in another field.

### *Syntax* How to Specify Data Validation

The syntax is

```
ACCEPT = list
ACCEPT = range
ACCEPT = FIND (sourcefield [AS targetfield] IN file)
```

where:

*list*
: Is a string of acceptable values. This is described in Chapter 4, *Describing Individual Fields.*

*range*
: Gives the range of acceptable values. This is described in Chapter 4, *Describing Individual Fields.*

FIND
: Verifies the incoming data against the values in an index in a FOCUS database.

*sourcefield*
: Is the name of the field to which the ACCEPT attribute is being applied or any other field in the same segment or path to the segment. This must be the actual field name, not the alias or a truncation of the name.

AS *targetfield*
: Is the name of the field that contains the acceptable data values. This field must be indexed.

IN *file*
: Is the name of the Master File describing the database that contains the indexed field of acceptable values.

# The INDEX Attribute

You can have FOCUS index the values of a field by including the INDEX attribute, or its alias of FIELDTYPE, in the field's declaration. An index is an internally stored and maintained table of data values and locations that speeds retrieval. You need to create an index if you want to:

- Join two segments. The cross–referenced field in a joined FOCUS database must be indexed, as described in *Describing Single Segments* on page 7-5 (for joins defined in a Master File), and the *Creating Reports* manual (for joins defined using the JOIN command).
- Create an alternate view and make it faster, as described in *Rotating a Database: Alternate Views* in Chapter 3, *Describing Groups of Fields*.
- Use a LOOKUP function in MODIFY.
- Use a FIND function in MODIFY.
- Speed segment selection and retrieval based on the values of a given field, as described for reporting in the *Creating Reports* manual.

## *Syntax* How to Specify Field Indexing

The syntax of the INDEX attribute in the Master File is:

```
INDEX = I or FIELDTYPE = I
```

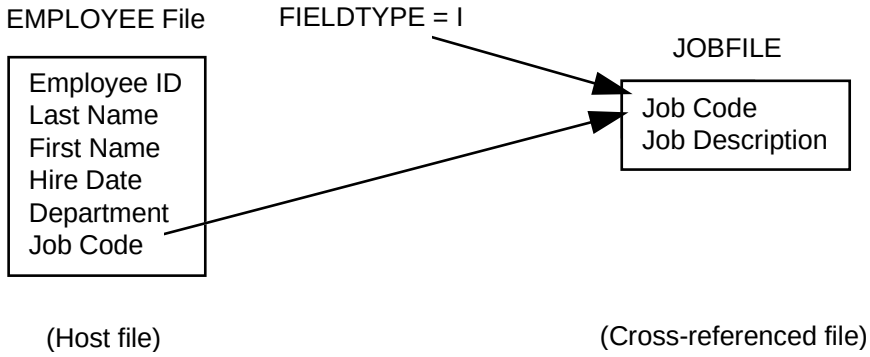
Text fields cannot be indexed. The maximum field name length for indexed fields is 12 characters.

For example:

```
FIELDNAME = JOBCODE, ALIAS = CJC, FORMAT = A3, INDEX = I, $
```

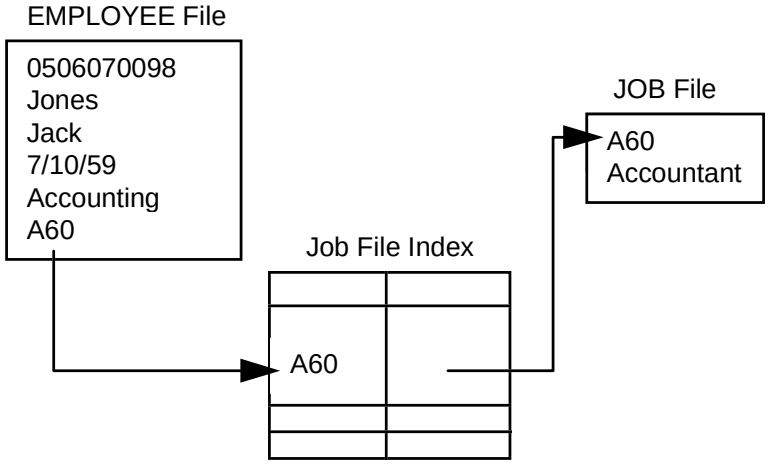
## Joins and the Index Attribute

In order for a segment to be cross-referenced with either a static cross-reference, a dynamic cross-reference, or a JOIN, at least one field in the cross-referenced segment must have INDEX=I or FIELDTYPE=I. This field, called the cross-referenced field, shares values with a field in the host file. Only the cross-referenced segment needs to have an indexed field, shown as follows:




Other files locate and use segments through these indexes. Any number of fields may be indexed on a segment, although it is advisable to limit the number of fields you index in a file.

FOCUS matches the value for the field named JOBCODE in the EMPLOYEE file to the field named JOBCODE in the JOBFILE file by using the index for the JOBCODE field in the JOBFILE file, as follows:



Indices are stored and maintained as part of the FOCUS file. The presence of the index is crucial to the operation of the cross-referencing facilities. Any number of external sources may locate and thereby share a segment because of it. New files which have data items in common with indexed fields in existing files can be added at any time.

### Maximum Number of Indices

The combined total of indices, text fields and segments cannot exceed 189 (of which a maximum of 64 can be segments and text location files).

*Reference*

### Usage Notes for INDEX

Note the following rules when using the INDEX attribute:

- **Alias.** INDEX has an alias of FIELDTYPE.
- **Changes.** If the INDEX attribute is removed from a field, or assigned a value of blank, which is equivalent, FOCUS ceases to maintain (update) it. If you no longer need the index, after you remove the INDEX attribute use the REORG option of the REBUILD facility to recover space occupied by the index. REBUILD is described in the *Maintaining Databases* manual.

  If you wish to turn off indexing temporarily—for example, to load a large amount of data into the database quickly—you can remove the INDEX attribute before loading the data, and then use the REBUILD command with the INDEX option to create the index. This is known as post indexing the file.

  You can assign the INDEX attribute to a field—that is, index the field—after the database has already been created and populated with records by using the REBUILD facility with the INDEX option. A total of seven indices may be added to the database using REBUILD INDEX. After eight indices have been added to a database in this way, you must use the REORG option of the REBUILD facility before adding a ninth. The following diagnostic message is issued if you attempt this:

  ```
  (FOC720) THE NUMBER OF INDEXES ADDED AFTER FILE CREATION EXCEEDS 7
  ```

## FORMAT and MISSING: Internal Storage Requirements

Some application developers find it useful to know how different data types and values are represented and stored internally.

- Integer fields are stored as full-word (four byte) binary integers.
- Floating-point double-precision fields are stored as double-precision (eight byte) floating-point numbers.
- Floating-point single-precision fields are stored as single-precision (four byte) floating-point numbers.
- Packed-decimal fields are stored as standard packed decimal numbers with up to 31 digits.
- Date fields are stored as full-word (four byte) binary integers representing the difference between the specified date and the date format's base date of December 31, 1900.
- Alphanumeric fields are stored as characters in the specified number of bytes.
- Missing values are represented internally by a dot (.) for alphanumeric fields, and as the value -9998998 for numeric fields.

# 8 Defining a Join in a Master File

**Topics:**

- Static Joins Defined in the Master File: SEGTYPE = KU and KM
- Using Cross-Referenced Descendant Segments: SEGTYPE = KL and KLU
- Dynamic Joins Defined in the Master File: SEGTYPE = DKU and DKM
- Comparing Static and Dynamic Master File Defined Joins and the JOIN Command
- Joining to One Cross-Referenced Segment From Several Host Segments

You can describe a new relationship between any two segments that have at least one field in common by joining them. The underlying data structures remain physically separate, but FOCUS treats them as if they were part of a single structure from which you can report. This chapter describes how to define a join in a Master File for FOCUS databases, fixed-format sequential data files, and VSAM files. For information about whether you can define a join in a Master File to be used with other types of data sources see the appropiate Interface manual.

You can join two databases in the following ways:

- **Dynamically using the JOIN command.** The join lasts for the duration of the FOCUS session (or until you clear it during the session) and creates a temporary view of the data that includes all of the segments in both databases. You can also use the JOIN command to join two data sources of any type, including a FOCUS database to a non-FOCUS data source. The JOIN command is described in detail in the *Creating Reports* manual.
- **Statically within a Master File.** This method is helpful if you want to access the joined structure frequently: the link (pointer) information needed to implement the join is permanently stored and does not need to be retrieved for each record during each request, saving you time. Like a dynamic Master File defined join, it is always available and retrieves only the segments that you specify. See *Static Joins Defined in the Master File: SEGTYPE = KU and KM*, on page 8-2. This is supported for FOCUS databases only.
- **Dynamically within a Master File.** This method saves you the trouble of issuing the JOIN command every time you need to join the databases and gives you flexibility in choosing the segments that will be available within the joined structure. See *Dynamic Joins Defined in the Master File: SEGTYPE = DKU and DKM*, on page 8-14.

**Development Tip:** Some users find it helpful to prototype a database design first using dynamic joins—implemented by issuing the JOIN command or within the Master File—and, once the design is stable, to change the frequently-used joins to static joins defined in the Master File, accelerating database access. Static joins should be used when the target or cross-referenced file contents do not change. You can change dynamic joins to static joins by using the REBUILD facility, as described in the *Maintaining Databases* manual.

**Note:** Master File defined joins are sometimes referred to as cross-references.

# Static Joins Defined in the Master File: SEGTYPE = KU and KM

Static joins allow you to relate segments in different FOCUS databases permanently. You specify static joins in the Master File of the host database.

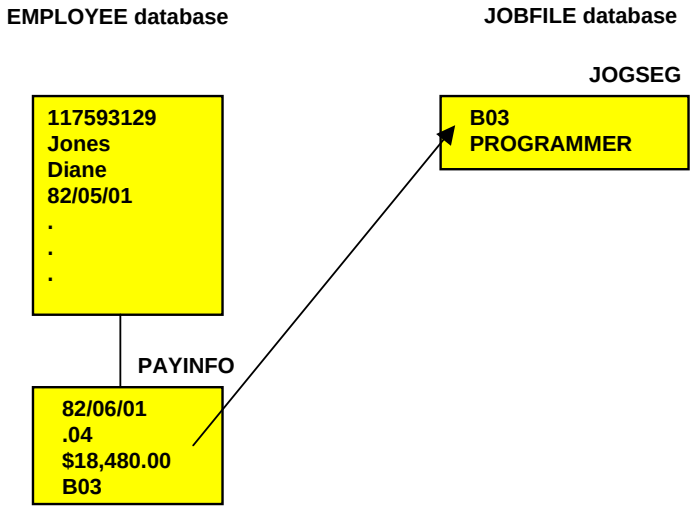
There are two types of static joins: one-to-one (SEGTYPE KU) and one-to-many (SEGTYPE KM).

- You specify a one-to-one join, also known as a unique join, when you want to retrieve at most one record instance from the cross-referenced database for each record instance in the host database.
- You specify a one-to-many join when you want to retrieve any number of record instances from the cross-referenced database.

## Describing a Unique Join: SEGTYPE = KU

In the EMPLOYEE database, there is a field named JOBCODE in the PAYINFO segment. The JOBCODE field contains a code that specifies the employee's job.

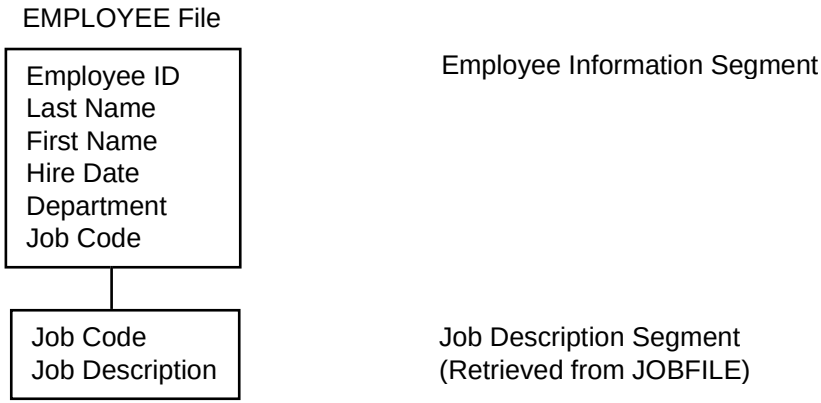
The complete description of the job and other related information is stored in a separate database named JOBFILE. You can retrieve the job description from JOBFILE by locating the record whose JOBCODE corresponds to the JOBCODE value in the EMPLOYEE database, as shown in the following diagram:




Using a join in this situation saves you the trouble of entering and revising the job description for every record in the EMPLOYEE database. Instead, you can maintain a single list of valid job descriptions in the JOBFILE database. Changes need be made only once, in JOBFILE, and are reflected in all of the corresponding joined EMPLOYEE database records.

Implementing the join as a static join is most efficient because the relationship between job codes and job descriptions is not likely to change.

Although the Employee Information and Job Description segments are stored in separate files, for reporting purposes FOCUS treats the EMPLOYEE file as though it also contains the Job Description segment from the JOBFILE file. The actual structure of the JOBFILE file is not affected. FOCUS will view the EMPLOYEE file as follows:




## *Syntax* How to Specify a Static Unique Join

You use the following syntax to create a static unique join:

```
SEGNAME = segname, SEGTYPE = KU, PARENT = parent,
CRFILE = db_name, CRKEY = field, [CRSEGNAME = crsegname,] $
```

where:

*segname*
: Is the name by which the cross-referenced segment will be known in the host database. You can assign any valid segment name, including the segment's original name in the cross-referenced database.

*parent*
: Is the name of the host segment.

*db_name*
: Is the name of the cross-referenced database. You can change the name without rebuilding the file.

*field*

Is the common name (field name and/or alias) of the host field and the cross-referenced field. The field name or alias of the host field must be identical to the field name of the cross-referenced field. You can change the field name without rebuilding the file as long as the SEGTYPE remains the same.

Both fields must have the same format type and length.

The cross-referenced field must be indexed (FIELDTYPE=I or INDEX=I).

*crsegname*

Is the name of the cross-referenced segment. If you do not specify this it defaults to the value assigned to SEGNAME. After data has been entered into the cross-referenced file, you cannot change the crsegname without rebuilding the file.

The SEGTYPE value KU stands for keyed unique.

## *Example* Creating a Static Unique Join

For example:

```
SEGNAME = JOBSEG, SEGTYPE = KU, PARENT = PAYINFO,
    CRFILE = JOBFILE, CRKEY = JOBCODE, $
```

The relevant sections of the EMPLOYEE Master File follow (for simplicity, fields and segments not essential to the example are not shown):

```
FILENAME = EMPLOYEE, SUFFIX = FOC, $

SEGNAME = EMPINFO, SEGTYPE = S1, $
    .
    .
    .
SEGNAME = PAYINFO, SEGTYPE = SH1, PARENT = EMPINFO, $
    FIELDNAME = JOBCODE,  ALIAS = JBC, FORMAT = A3, $
    .
    .
    .
SEGNAME = JOBSEG, SEGTYPE = KU, PARENT = PAYINFO, CRFILE = JOBFILE,
    CRKEY = JOBCODE, $
```

Note that you only have to give the name of the cross-referenced segment; the fields in that segment are already known from the cross-referenced database's Master File (JOBFILE in this example). Note that the CRSEGNAME attribute is omitted, since in this example it is identical to the name assigned to the SEGNAME attribute.

The Master File of the cross-referenced database, as well as the database itself, must be accessible whenever the host database is used. There does not need to be any data in the cross-referenced database.

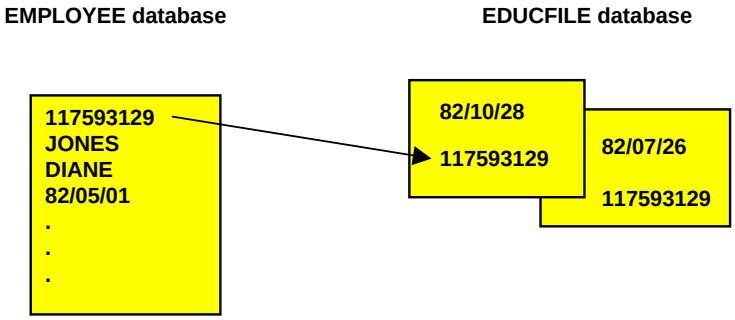
### Using a Unique Join for Decoding

Decoding is the process of matching a code (such as the job code in our example) to the information it represents (such as the job description). Because every code has only one set of information associated with it, the join between the code and the information should be one-to-one, that is, unique. You can decode using a join, as in our example, or using the DECODE function with the DEFINE FILE command, as described in the *Creating Reports* manual. The join method is recommended when there are a large number of codes.

## Describing a Non-Unique Join: SEGTYPE = KM

You use a one-to-many join (that is, a non-unique join) when you have several instances of data in the cross-referenced segment associated with a single instance in the host segment. Using our EMPLOYEE example, suppose that you kept an educational database named EDUCFILE to track the course work employees were doing. One segment in that database, ATTNDSEG, contains the dates on which each employee attended a given class. The segment is keyed by attendance date. The EMP_ID field, which identifies the attendees, contains the same ID numbers as the EMP_ID field in the EMPINFO segment of the EMPLOYEE database.

If you want to see an employee's educational record, you can join the EMP_ID field in the EMPINFO segment to the EMP_ID field in the ATTNDSEG segment. You should make this a one-to-many join, since you want to retrieve all instances of class attendance associated with a given employee ID:



*Syntax*

## How to Specify a Static Multiple Join

The syntax for describing one-to-many joins is similar to that for one-to-one joins described in *How to Specify a Static Unique Join*, on page 8-4, except that you supply a different value, KM (which stands for keyed multiple), for the SEGTYPE attribute, as follows:

```
SEGTYPE = KM
```

*Example*

## Specifying a Static Multiple Join

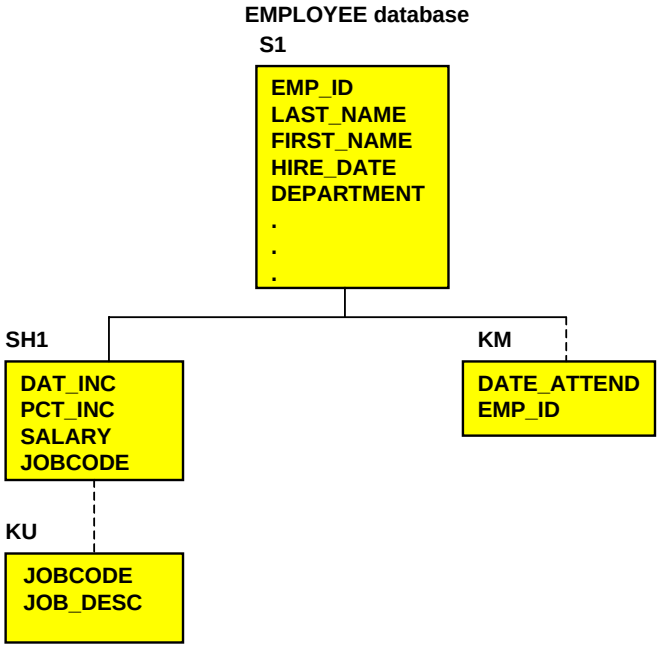
For example:

```
SEGNAME = ATTNDSEG, SEGTYPE = KM, PARENT = EMPINFO,
    CRFILE = EDUCFILE, CRKEY = EMP_ID, $
```

The relevant sections of the EMPLOYEE Master File follow (nonessential fields and segments are not shown):

```
FILENAME = EMPLOYEE, SUFFIX = FOC, $

SEGNAME = EMPINFO, SEGTYPE = S1, $
    FIELDNAME = EMP_ID, ALIAS = EID, FORMAT = A9, $
    .
    .
    .
SEGNAME = PAYINFO, SEGTYPE = SH1, PARENT = EMPINFO, $
    FIELDNAME = JOBCODE, ALIAS = JBC, FORMAT = A3, $
    .
    .
    .
SEGNAME = JOBSEG, SEGTYPE = KU, PARENT = PAYINFO,  CRFILE = JOBFILE,
    CRKEY = JOBCODE, $
    .
    .
    .
SEGNAME = ATTNDSEG, SEGTYPE = KM, PARENT = EMPINFO, CRFILE = EDUCFILE,
    CRKEY = EMP_ID, $
```

Within a report request, FOCUS treats both cross-referenced databases, JOBFILE and EDUCFILE, as though they are part of the EMPLOYEE database. The data structure resembles the following:



# Using Cross-Referenced Descendant Segments: SEGTYPE = KL and KLU

When you join two databases, you can access any or all of the segments in the cross-referenced database, not just the cross-referenced segment itself. These other segments are sometimes called linked segments. From the perspective of the host database, all of the linked segments are descendants of the cross-referenced segment; it is as though an alternate view had been taken on the cross-referenced database to make the cross-referenced segment the root. To access a linked segment, you only need to declare it in the Master File of the host database.

***Syntax*** **How to Identify Cross-Referenced Descendant Segments**

The syntax is:

```
SEGNAME = segname, SEGTYPE = {KL|KLU}, PARENT = parentname,
    CRFILE = db_name, [CRSEGNAME = crsegname,] $
```

where:

*segname*
: Is the name assigned to the cross-referenced segment in the host database.

KL
: Indicates that this segment is a descendant segment in a cross-referenced database (as viewed from the perspective of the host database), and has a one-to-many relationship to its parent. KL stands for keyed through linkage.

KLU
: Indicates that this segment is a descendant segment in a cross-referenced database (as viewed from the perspective of the host database), and has a one-to-one relationship to its parent. KLU stands for keyed through linkage, unique.

*parentname*
: Is the name of the segment's parent in the cross-referenced database, as viewed from the perspective of the host database.

*db_name*
: Is the name of the cross-referenced database. You can change the name without rebuilding the file.

*crsegname*
: Is the name of the cross-referenced segment. If you do not specify this it defaults to the value assigned to SEGNAME.

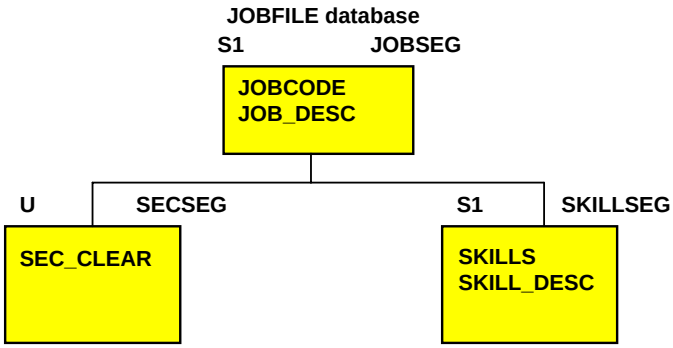
For example:

```
SEGNAME = SECSEG, SEGTYPE = KLU, PARENT = JOBSEG, CRFILE = JOBFILE, $
SEGNAME = SKILLSEG, SEGTYPE = KL, PARENT = JOBSEG, CRFILE = JOBFILE, $
```

Note that you do not use the CRKEY attribute in a declaration for a linked segment, since the common join field (which is identified by CRKEY) only needs to be specified for the cross-referenced segment.

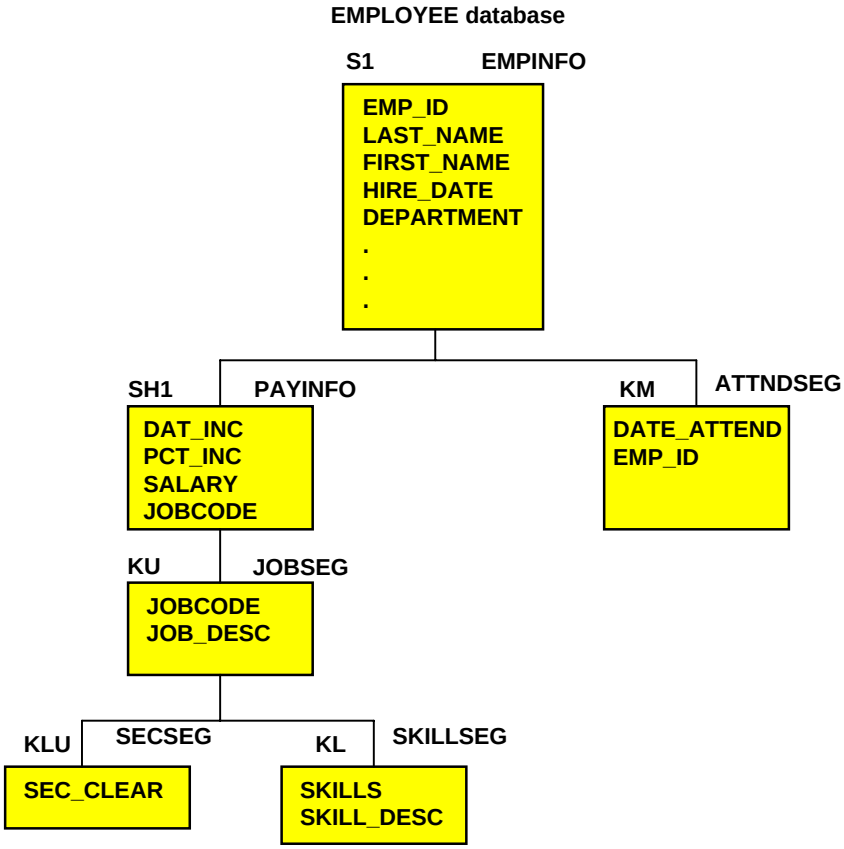
### *Example* Using Cross-Referenced Descendant Segments

Consider our EMPLOYEE example. JOBFILE is a multi-segment database:




In your EMPLOYEE database application, you may need the security information stored in the SECSEG segment and the job skill information stored in the SKILLSEG segment. Once you have created a join, you can access any or all of the other segments in the cross-referenced database using the SEGTYPE value KL for a one-to-many relationship (as seen from the host database), and KLU for a one-to-one relationship (as seen from the host database). KL and KLU are used to access descendant segments in a cross-referenced database for both static (KM) and dynamic (DKM) joins.

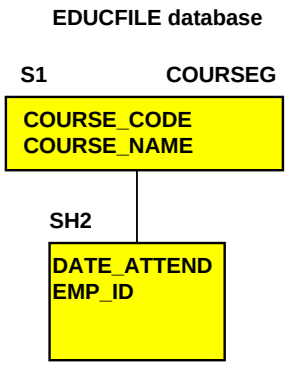
When FOCUS retrieves the JOBSEG segment from JOBFILE, it also retrieves all of JOBSEG's children that were declared with KL or KLU SEGTYPEs in the EMPLOYEE Master File:



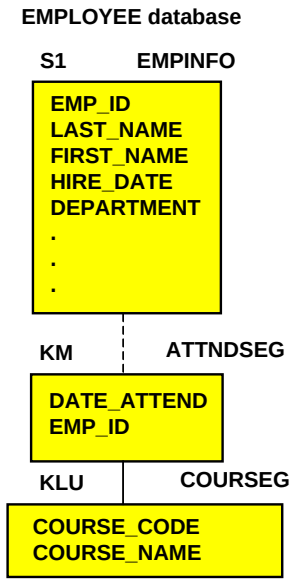
## *Example* Using Cross-Referenced Ancestral Segments

Remember that you can retrieve all of the segments in a cross-referenced database, including both descendants and ancestors of the cross-referenced segment. Ancestor segments should be declared in the host Master File with a SEGTYPE of KLU, as a segment can have only one parent and so, from the perspective of the host database, this is a one-to-one relationship.

Consider the EDUCFILE database used in our example. The COURSEG segment is the root and describes each course; ATTNDSEG is a child and includes employee attendance information:




When you join EMPINFO in EMPLOYEE to ATTNDSEG in EDUCFILE, you can access course descriptions in COURSEG by declaring it as a linked segment. From this perspective, COURSEG is a child of ATTNDSEG:

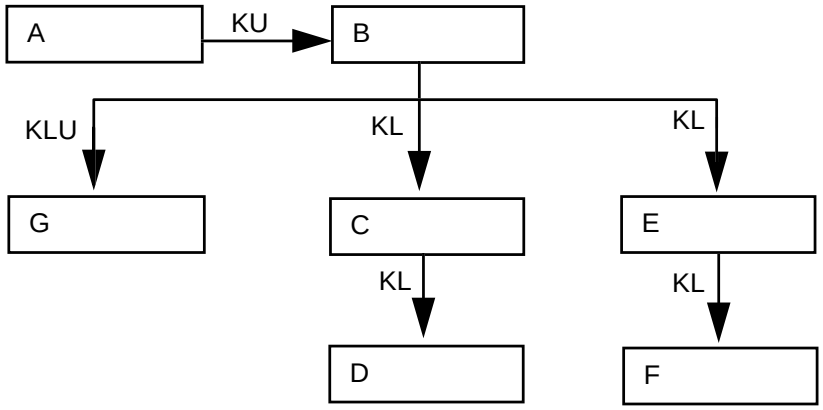


The sections of the EMPLOYEE Master File used in our examples follow (nonessential fields and segments are not shown):

```
FILENAME = EMPLOYEE, SUFFIX = FOC, $

SEGNAME = EMPINFO, SEGTYPE = S1, $
    FIELDNAME = EMP_ID, ALIAS = EID, FORMAT = A9, $
    .
    .
    .
SEGNAME = PAYINFO, SEGTYPE = SH1, PARENT = EMPINFO, $
    FIELDNAME = JOBCODE, ALIAS = JBC, FORMAT = A3, $
    .
    .
    .
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, $
```

## Hierarchies of Linked Segments

A KL segment may lead to other KL segments. Graphically, this can be illustrated as:




The letters on the arrows are the SEGTYPEs.

Note that segment G may either be a unique descendant of B or B's parent.

# Dynamic Joins Defined in the Master File: SEGTYPE = DKU and DKM

You can define a dynamic join in a Master File using the SEGTYPE attribute. There are two types of dynamic Master File defined joins: one-to-one (SEGTYPE DKU) and one-to-many (SEGTYPE DKM).

- As with a static join, you specify a one-to-one join, also known as a unique join, when you want to retrieve at most one record instance from the cross-referenced database for each record instance in the host database.
- You specify a one-to-many join when you want to retrieve any number of record instances from the cross-referenced database.

The difference between static and dynamic joins has to do with storage, speed, and flexibility:

- The links (pointers) for a static join are retrieved once and then permanently stored in the host database (and automatically updated as needed).
- The links for a dynamic join are not saved and need to be retrieved for each record in each report request.

This makes static joins much faster than dynamic ones, but harder to change: you can only redefine or remove a static join using the REBUILD facility, as described in the *Maintaining Databases* manual. You can redefine or remove a dynamic join at any time by editing the Master File.

***Syntax*** **How to Specify a Dynamic Join**

You specify a dynamic Master File defined join the same way that you specify a static join (as described in *How to Specify a Static Unique Join*, on page 8-4), except that the value of the SEGTYPE attribute for the cross-referenced segment is DKU (standing for dynamic keyed unique) for a one-to-one join, and DKM (standing for dynamic keyed multiple) for a one-to-many join.

For example:

```
SEGNAME = JOBSEG, SEGTYPE = DKU, PARENT = PAYINFO,
    CRFILE = JOBFILE, CRKEY = JOBCODE, $
```

You declare linked segments in a dynamic join the same way that you do in a static join. In both cases SEGTYPE has a value of KLU for unique linked segments, and KL for non-unique linked segments.

***Example*** **Specifying a Dynamic Join**

The following Master File includes the relevant sections of EMPLOYEE and the segments joined to it, but with the static joins replaced by dynamic joins (nonessential fields and segments are not shown):

```
FILENAME = EMPLOYEE, SUFFIX = FOC, $

SEGNAME = EMPINFO, SEGTYPE = S1, $
    FIELDNAME = EMP_ID,        ALIAS = EID, FORMAT = A9, $
    .
    .
    .
SEGNAME = PAYINFO, SEGTYPE = SH1,  PARENT = EMPINFO, $
    FIELDNAME = JOBCODE,       ALIAS = JBC, FORMAT = A3, $
    .
    .
    .
SEGNAME = JOBSEG,  SEGTYPE = DKU,  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 = DKM,  PARENT = EMPINFO, CRFILE = EDUCFILE,
    CRKEY = EMP_ID, $

SEGNAME = COURSEG, SEGTYPE = KLU,  PARENT = ATTNDSEG,CRFILE = EDUCFILE, $
```

# Comparing Static and Dynamic Master File Defined Joins and the JOIN Command

If you wish to join two FOCUS databases, you can choose between two types of joins (static and dynamic) and two methods of defining the join (defined in the Master File and defined by issuing the JOIN command).

- For a static join, the links, which point from a host segment instance to the corresponding cross-referenced segment instance, are created once and then permanently stored and automatically maintained in the host database.
- For a dynamic join, the links are retrieved each time they are needed. This makes static joins faster than dynamic ones, since the links only need to be established once, but less flexible, as you can only redefine or remove a static join by using the REBUILD facility. The REBUILD facility is described in the *Maintaining Data* manual.

Among dynamic joins the JOIN command is easier to use in that you do not need to edit the Master File each time you want to change the join specification, and you do not need to describe each linked segment as it appears from the perspective of the host database. On the other hand, Master File defined dynamic joins enable you to omit unnecessary cross-referenced segments.

You may find it efficient to implement frequently-used joins as static joins. You can change static joins to dynamic, and dynamic to static, using the REBUILD facility.

The following chart compares implementing a static join defined in a Master File, a dynamic join defined in a Master File, and a dynamic join defined by issuing the JOIN command.

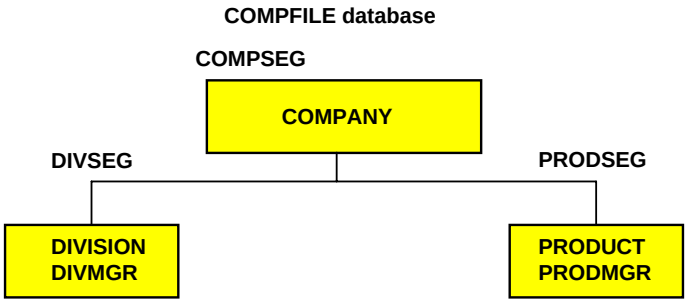
| | **Advantages** | **Disadvantages** |
|---|---|---|
| **Static Join in Master File**<br><br>**(SEGTYPE = KU or KM)** | Faster after first use: links are created only once.<br><br>Always in effect.<br><br>Can select some linked segments and omit others. | Must be specified before database is created or reloaded using REBUILD.<br><br>Requires REBUILD utility to change.<br><br>Requires four bytes of file space per instance.<br><br>User needs to know how to specify relationships for linked segments (KL, KLU). |
| **Dynamic Join in Master File**<br><br>**(SEGTYPE = DKU or DKM)** | Can be specified at any time.<br><br>Always in effect. Does not use any space in the database.<br><br>Can be changed or removed as needed, without using the REBUILD facility.<br><br>Can select some linked segments and omit others. | Slower: links are retrieved for each record in each report request.<br><br>User needs to know how to specify relationships for linked segments (KL, KLU). |
| **Dynamic Join (using the JOIN Command)** | Can be specified at any time.<br><br>Does not use any space in the database. Can be changed or removed as needed, without using the REBUILD facility.<br><br>User never needs to describe relationships of linked segments. | Slower: links are retrieved for each record in each report request.<br><br>JOIN command must be issued in each session in which you want the join to be in effect.<br><br>All linked segments are always included, whether or not you need them. |

# Joining to One Cross-Referenced Segment From Several Host Segments

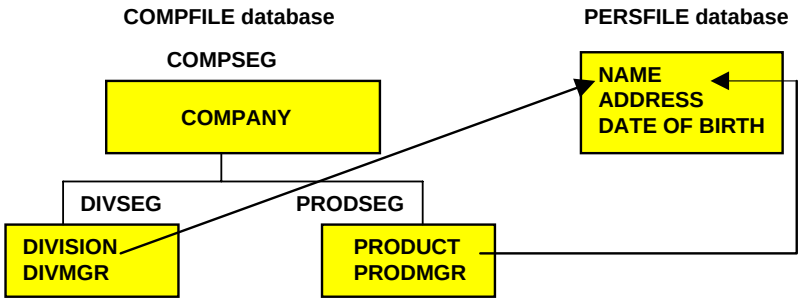
You may come upon situations where you need to join to one cross-referenced segment from several different segments in the host database. You may also find a need to join to one cross-referenced segment from two different host databases at the same time. You can handle these data structures using Master File defined joins.

## Joining From Several Segments in One Host Database

In an application, you may want to use the same cross-referenced segment in several places in the same database. Suppose, for example, that you have a database named COMPFILE that maintains data on companies you own:




The DIVSEG segment contains an instance for each division and includes fields for the name of the division and its manager. Similarly, the PRODSEG segment contains an instance for each product and the name of the product manager. You might want to retrieve personal information for both the product managers and the division managers from a single personnel database, as shown below:



You cannot retrieve this information with a standard Master File defined join because there are two cross-reference keys in the host file (PRODMGR and DIVMGR) and in your reports you will want to distinguish addresses and dates of birth retrieved for the PRODSEG segment from those retrieved for the DIVSEG segment.

FOCUS provides a way for you to implement a join to the same cross-referenced segment from several segments in the one host database: you can match the cross-referenced and host fields from alias to field name and uniquely rename the fields.

The Master File of the PERSFILE might look like this:

```
FILENAME = PERSFILE, SUFFIX = FOC, $
SEGNAME = IDSEG, SEGTYPE = S1, $
    FIELD = NAME,        ALIAS = FNAME, FORMAT = A12,          INDEX=I,     $
    FIELD = ADDRESS,     ALIAS = DAS,   FORMAT = A24,                       $
    FIELD = DOB,         ALIAS = IDOB,  FORMAT = YMD,                       $
```

You use the following Master File to join PERSFILE to COMPFILE. Note that there is no record terminator ($) following the cross-referenced segment declaration (preceding the cross-referenced field declarations).

```
FILENAME = COMPFILE, SUFFIX = FOC, $

SEGNAME = COMPSEG, SEGTYPE = S1, $
            FIELD = COMPANY,   ALIAS = CPY,   FORMAT = A40,   $

SEGNAME = DIVSEG,  PARENT = COMPSEG, SEGTYPE = S1, $
            FIELD = DIVISION,  ALIAS = DV,      FORMAT = A20,            $
            FIELD = DIVMGR,    ALIAS = NAME,    FORMAT = A12,            $

SEGNAME = ADSEG,   PARENT = DIVSEG,  SEGTYPE = KU,
            CRSEGNAME = IDSEG, CRKEY = DIVMGR, CRFILE = PERSFILE,
            FIELD = NAME,      ALIAS = FNAME,   FORMAT = A12, INDEX = I, $
            FIELD = DADDRESS,  ALIAS = ADDRESS, FORMAT = A24,            $
            FIELD = DDOB,      ALIAS = DOB,     FORMAT = YMD,            $

SEGNAME = PRODSEG, PARENT = COMPSEG, SEGTYPE = S1, $
            FIELD = PRODUCT,   ALIAS = PDT,     FORMAT = A8,             $
            FIELD = PRODMGR,   ALIAS = NAME,    FORMAT = A12,            $

SEGNAME = BDSEG,   PARENT = PRODSEG, SEGTYPE = KU,
            CRSEGNAME = IDSEG, CRKEY = PRODMGR, CRFILE = PERSFILE,
            FIELD = NAME,      ALIAS = FNAME,   FORMAT = A12, INDEX = I, $
            FIELD = PADDRESS,  ALIAS = ADDRESS, FORMAT = A24,            $
            FIELD = PDOB,      ALIAS = DOB,     FORMAT = YMD,            $
```

DIVMGR and PRODMGR are described as CRKEYs. FOCUS automatically matches their common alias, NAME, to the field name NAME in the PERSFILE database. In addition, the field declarations that follow the join information rename the ADDRESS and DOB fields so that they can be referred to separately in reports. Their actual field names in the PERSFILE are supplied as aliases.

Note that the NAME field cannot be renamed, since it is the common join field. It must be included in the declaration along with the fields being renamed, as it is described in the cross-referenced database. That it cannot be renamed is not a problem, since its ALIAS can be renamed, and, in any event, the field does not need to be used in reports: because it is the join field, it contains exactly the same information as the DIVMGR and PRODMGR fields.

The following conventions must be observed:

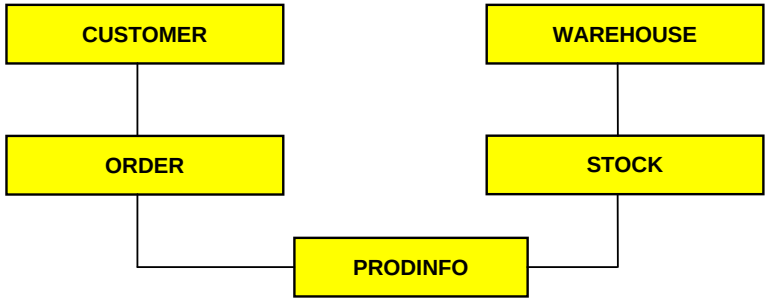
- The common join field's FIELDNAME or ALIAS in the host database must be identical to its FIELDNAME in the cross-referenced database.
- The common join field should not be renamed, but the alias can be changed. The other fields in the cross-referenced segment can be renamed.
- Place field declarations for the cross-referenced segment after the cross-referencing information in the Master File of the host file, in the order in which they actually occur in the cross-referenced segment. Omit the record terminator ($) at the end of the cross-referenced segment declaration in the host Master File, as shown:

```
SEGNAME = BDSEG, PARENT = PRODSEG, SEGTYPE = KU,
    CRSEGNAME = IDSEG, CRKEY = PRODMGR, CRFILE = PERSFILE,
    FIELD = NAME,     ALIAS = FNAME,   FORMAT = A12 ,INDEX=I, $
    FIELD = PADDRESS, ALIAS = ADDRESS, FORMAT = A24         , $
    FIELD = PDOB,     ALIAS = DOB,     FORMAT = YMD         , $
```

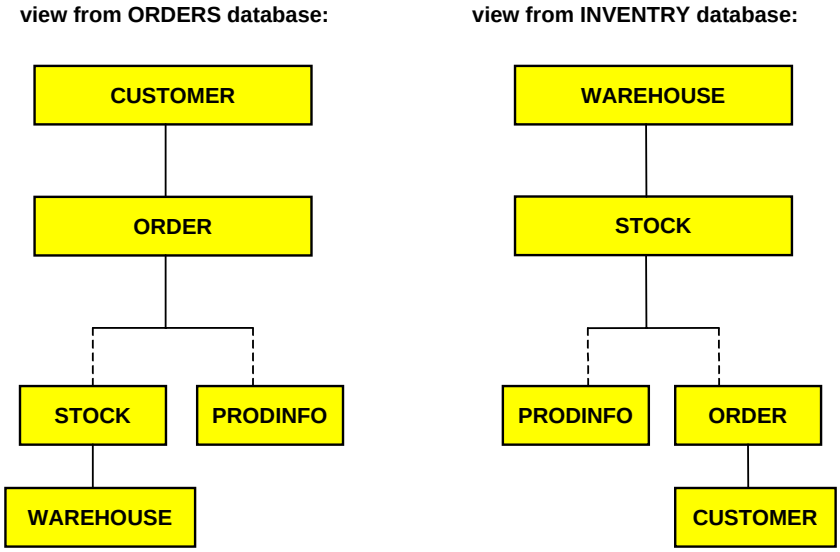
# Joining From Several Segments in Several Host Databases: Multiple Parents

At some point you may need to join to a cross-referenced segment from two different host databases at the same time. If you were to describe a structure like this as a single database, you would have to have two parents for the same segment, which is invalid. You can, however, describe the information in separate databases, using joins to achieve a similar effect.

Consider an application that keeps track of customer orders for parts, warehouse inventory of parts, and general part information. If this were described as a single database, it would be structured as follows:




You can join several databases to create this structure. For example:



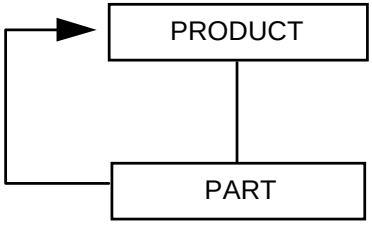
The CUSTOMER and ORDER segments are in the ORDERS database, the WAREHOUSE and STOCK segments are in the INVENTRY database, and the PRODINFO segment is stored in the PRODUCTS database. Both the INVENTRY and ORDERS databases have one-to-one joins to the PRODUCTS database. In the INVENTRY database, STOCK is the host segment; in the ORDERS database, ORDER is the host segment.

In addition, there is a one-to-many join from the STOCK segment in the INVENTRY database to the ORDER segment in the ORDERS database, and a reciprocal one-to-many join from the ORDER segment in the ORDERS database to the STOCK segment in the INVENTRY database.

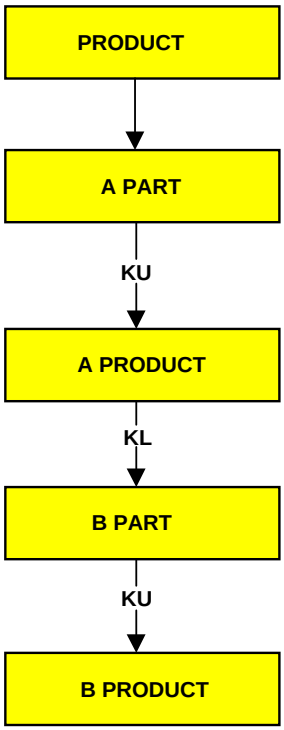
The joins among these three databases can be viewed from the perspectives of both host databases, approximating the multiple-parent structure described earlier.

## Recursive Reuse of a Segment

In rare cases a file may cross-reference itself. Consider the case of a file of products, each with a list of parts that compose the product, where a part may itself be a product and have sub-parts. Schematically, this would appear as:



A FOCUS description for this case, shown for two levels of sub-parts, is:




See the *Creating Reports* manual for more information on recursive joins.

# 9 Checking and Changing Master Files: CHECK

**Topics:**

- CHECK Command Display
- The PICTURE Option
- The HOLD Option

Use the CHECK command to validate your Master Files. You must always do this after writing the Master File. If you do not issue the CHECK command, FOCUS may not update your Master File with the changes that you just made. The CHECK output highlights any errors in your Master File and allows you to correct them before reading the data source. After making any necessary corrections, use CHECK again to confirm that the Master File is valid.

## *Syntax* How to Check File Descriptions

The syntax of the CHECK command is

```
CHECK FILE filename[.field] [PICTURE [RETRIEVE]] [DUPLICATE]
[HOLD [AS name][ALL]]
```

where:

*filename*
Is the name under which you created the Master File.

*.field*
Is used for an alternate view of the Master File.

PICTURE
Is an option that displays a diagram showing the complete file structure. The keyword PICTURE can be abbreviated to PICT. This option is explained in *The PICTURE Option*, on page 9-5.

RETRIEVE
Alters the picture to reflect the order in which segments are retrieved when TABLE or TABLEF commands are issued. Note that unique segments are viewed as logical extensions of their parent segment. The keyword RETRIEVE can be abbreviated to RETR.

DUPLICATE
Lists duplicate field names for the specified file. The keyword DUPLICATE can be abbreviated to DUPL.

HOLD
Generates a temporary HOLD file and HOLD Master File containing information about fields in the file. You can use this HOLD file to write reports. The AS option specifies a field name for your files. The option is described and illustrated in *The HOLD Option*, on page 9-7.

*name*
Is a name for the HOLD file and HOLD Master File.

ALL
Adds the values of FDEFCENT and FYRTHRESH at the file level and the values of DEFCENT and YRTHRESH at the field level to the HOLD file.

# CHECK Command Display

If your Master File contains syntactical errors, the CHECK command displays appropriate error messages.

If the file description has no syntactical errors, the CHECK command displays the following message

```
NUMBER OF ERRORS=             0
NUMBER OF SEGMENTS=           n ( REAL=      n VIRTUAL=     n )
NUMBER OF FIELDS=             n INDEXES=     n FILES=       n
NUMBER OF DEFINES=            n
TOTAL LENGTH OF ALL FIELDS = n
```

where:

`NUMBER OF ERRORS`
: Indicates the number of syntactical errors in the Master File.

`NUMBER OF SEGMENTS`
: Is the number of segments in the Master File, including cross-referenced segments.

`REAL`
: Is the number of segments that are not cross-referenced. These segments have types Sn, SHn, U, or blank.

`VIRTUAL`
: Is the number of segments that are cross-referenced. These segments have types KU, KLU, KM, KL, DKU, or DKM.

`NUMBER OF FIELDS`
: Is the number of fields described in the Master File.

`INDEXES`
: Is the number of indexed fields. These fields have the attribute FIELDTYPE=I or INDEX=I in the Master File.

`FILES`
: Is the number of data files containing the fields.

`NUMBER OF DEFINES`
: Is the number of DEFINE fields in the Master File. This message displays only if DEFINE fields are specified.

`TOTAL LENGTH`
: Is the total length of all fields as defined in the Master File by either the FORMAT attribute (if the data file is a FOCUS file) or the ACTUAL attribute (if the data file is an external file).

### *Example* Using the CHECK File Command

For example, entering the following command

```
CHECK FILE EMPLOYEE
```

produces the following information:

```
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
```

When you are using FOCUS online, this message appears on the terminal even if the PRINT parameter is set to OFFLINE.

## Determining Common Errors

- If the data file is an external file, check the TOTAL LENGTH OF ALL FIELDS that appears near the top of your screen to verify the accuracy of the field lengths you have specified for the file. One of the most common causes of errors in generating reports from external files is incorrectly specified field lengths. The number given as the total length of all fields should be equal to the logical record length of the external file.

  In general, if the total length of all fields is not equal to the logical record length of the external file, you have specified the length of at least one field incorrectly. Your external data may not be read correctly if you do not correct the error.

- If the following warning message is generated

  ```
  (FOC1829) WARNING. FIELDNAME IS NOT UNIQUE WITHIN A SEGMENT: fieldname
  ```

  it is because duplicate fields (those having the same field names and aliases) are not allowed in the same segment. The second occurrence is never accessed by FOCUS.

  When the CHECK command is issued for a file that has more than one field of the same name within the same segment, a FOC1829 error message is generated along with a warning indicating the duplicate field names, such as the following:

  ```
  (FOC1829) WARNING. FIELDNAME IS NOT UNIQUE WITHIN A SEGMENT: BB
  WARNING: FOLLOWING FIELDS CANNOT BE ACCESSED
  BB                                      IN SEGMENT SEGA       (VS SEGB )
  ```

  When the DUPLICATE option is added, the output contains a warning message like the following:

  ```
  WARNING: FOLLOWING FIELDS APPEAR MORE THAN ONCE
  AA                                      IN SEGMENT SEGB       (VS SEGA )
  ```

# The PICTURE Option

The PICTURE option displays a diagram of the FOCUS structure defined by the Master File. Each segment is represented by a box. There are four types of boxes, which indicate whether a segment (including the root segment) is non-unique or unique and whether it is real or cross-referenced. The four types of boxes are:

**Real segments**

**Non-unique segment:**

```
            segname
num         segtype
**************
*field1        **I
*field2        **
*field3        **
*field4        **
*              **
***************
 **************
```

**Unique segment:**

```
            segname
num         U
**************
*field1        *I
*field2        *
*field3        *
*field4        *
*              *
**************
```

**Cross-referenced segments**

**Non-unique segment:**

```
            segname
num         KM (or KLM)
..............
:field1        ::K
:field2        ::
:field3        ::
:field4        ::
:              ::
:............::
 ............:
        crfile
```

**Unique segment**

```
            segname
num         KU (or KLU)
..............
:field1        :K
:field2        :
:field3        :
:field4        :
:              :
:............:
        crfile
```

where:

*num*

Is the number assigned to the segment in the FOCUS structure.

*segname*

Is the name of the segment.

*segtype*

Is the segment type for a real, non-unique segment: Sn, SHn, or N (for blank segtypes).

*field1 ...*

Are the names of fields in the segment. Field names of 66 characters are truncated to 12 characters in CHECK FILE PICTURE operations.

`I`
: Indicates an indexed field.

`K`
: Indicates the key field in the cross-referenced segment.

*`crfile`*
: Is the name of the cross-referenced file if the segment is cross-referenced.

The diagram also shows the relationship between segments (see the following example). Parent segments are shown above children segments connected by straight lines.

## *Example* Using the CHECK FILE PICTURE Option

The following diagram shows the structure of the JOB file joined to the SALARY file:

```
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 02/08/99 AT 12.33.04

         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 HOLD Option

The HOLD option generates a temporary HOLD file. HOLD files are explained in the *Creating Reports* manual. This HOLD file contains detailed information regarding file, segment, and field attributes, which you can display in reports using TABLE requests.

Certain fields in this HOLD file are of special interest. Unless otherwise noted, these fields are named the same as attributes in Master Files; each field stores the values of the similarly-named attribute. The fields can be grouped into file attributes, segment attributes, and field attributes.

**File Attributes:**

`FILENAME`

`SUFFIX`

`FDEFCENT, FYRTHRESH`

Note that these attributes are included in the HOLD file, if they exist in the original Master File and you specify the ALL option.

**Segment Attributes:**

`SEGNAME`

`SEGTYPE`

Note that this field does not indicate the number of segment key fields. Segment types S1, S2, and so on are shown as type S. The same is true with segment type SHn.

`SKEYS`

The number of segment key fields. For example, if the segment type is S2, SKEYS has the value 2.

`SEGNO`

The number assigned to the segment within the FOCUS structure. This is displayed in the picture.

`LEVEL`

The level of the segment within the FOCUS structure. The root segment is on Level 1, its children are on Level 2, and so on.

`PARENT`

`CRKEY`

`FIELDNAME`

**Field Attributes:**

ALIAS

FORMAT

ACTUAL

Note that if you include the FORMAT field in the TABLE request, you should not use the full field name FORMAT. Rather, you should use the alias USAGE or a unique truncation of the FORMAT field name (the shortest unique truncation is FO).

DEFCENT, YRTHRESH

Note that these attributes are included in the HOLD file, if they exist in the original Master File and you specify the ALL option.

## *Example* Using the CHECK FILE HOLD Option

This sample FOCUS procedure creates a HOLD file describing the EMPLOYEE file. It then writes a report that displays the names of cross-referenced segments in the EMPLOYEE file, their segment types, and the attributes of their fields: field names, aliases, and formats.

```
CHECK FILE EMPLOYEE HOLD
TABLE FILE HOLD
HEADING
"FIELDNAMES, ALIASES, AND FORMATS"
"OF CROSS-REFERENCED FIELDS IN THE EMPLOYEE FILE"
" "
PRINT FIELDNAME ALIAS USAGE BY SEGNAME BY SEGTYPE
WHERE SEGTYPE CONTAINS 'K'
END
```

The request produces this report:

```
PAGE 1

FIELDNAMES, ALIASES, AND FORMATS
OF CROSS-REFERENCED FIELDS IN THE EMPLOYEE FILE

SEGNAME    SEGTYPE            FIELDNAME      ALIAS   FORMAT
-------    ------             ---------      -----   ------
ATTNDSEG   KM                 DATE_ATTEND    DA      I6YMD
                              EMP_ID         EID     A9
COURSEG    KLU                COURSE_CODE    CC      A6
                              COURSE_NAME    CD      A30
JOBSEG     KU                 JOBCODE        JC      A3
                              JOB_DESC       JD      A25
SECSEG     KLU                SEC_CLEAR      SC      A6
SKILLSEG   KL                 SKILLS                 A4
                              SKILL_DESC     SD      A30
```

### *Example* Using the CHECK FILE HOLD ALL Option

Assume the Employee database contains the following FILE declaration:

```
FILENAME = EMPLOYEE, SUFFIX = FOC, FDEFCENT = 19, FYRTHRESH = 50
```

The following request:

```
CHECK FILE EMPLOYEE HOLD ALL
TABLE FILE HOLD
PRINT FDEFCENT FYRTHRESH
END
```

produces the following report:

```
FDEFCENT   FYRTHRESH
--------   ---------

      19          50
```

## Specifying AS Names With the HOLD Option

An AS name may be provided for the temporary HOLD file generated by the CHECK command. If a name is not specified, the default name is HOLD and FOCUS will replace any existing default file.

**Note:**

- The AS name may not be longer than 8 characters, or FOCUS defaults to the name HOLD and no warning is issued.
- When the AS option is specified in combination with other CHECK options, the AS holdname specification must appear last.

## TITLE, HELPMESSAGE, and TAG Attributes

When you use the HOLD option of the CHECK command, FOCUS places the TITLE text in the TITLE field of the FLDATTR segment, the HELPMESSAGE text in the HELPMESSAGE field of the FLDATTR segment, and the TAG names in the TAGNAME field of the SEGATTR segment.

When no JOINs are in effect, or when a JOIN command is issued without a TAG name, the TAGNAME field by default contains the name of the file specified in the CHECK command. When JOINs are issued in conjunction with the TAG name feature, the TAGNAME field contains the TAG name for the host and cross-referenced files.

**Note:** The addition of these three fields caused an increase in the LRECL of the HOLD FOCTEMP file to 370 when SET FIELDNAME=NEW and to 238 when SET FIELDNAME=OLD.

## DEFINE Fields in the Master File

With the HOLD option, defined fields are placed in the segment in which they would be stored if they were real fields in the file. This is not necessarily the physical location of the field in the Master File, but the lowest segment that must be accessed in order to evaluate the expression defining the field. Fields whose values are not dependent on retrieval default to the top segment. The value of FLDSEG in the FLDATTR segment is zero for these fields. The format of FLDSEG is I2S in the Master File, which causes zero to be displayed as blank in reports. FLDSEG may be dynamically reformatted in a TABLE request (FLDSEG/I2) to force the display of zero.

Once data has been entered into a FOCUS file you can no longer make arbitrary changes to the Master File. Some changes are entirely harmless and can be made at any time; others are prohibited unless the data is reentered or the file rebuilt. A few others can be made if corresponding changes are made in several places.

You can use a system editor or TED to make permitted changes to the file containing the Master File. The checking procedure, CHECK, should be used after any change.

# 10 Accessing FOCUS Databases: USE

**Topics:**

- Using Alternative File Specifications
- Identifying New Databases to FOCUS
- Protecting Files (Read Only)
- Concatenating Databases
- Specifying Databases in Simultaneous Usage Mode
- Using the LOCATION Attribute
- Displaying the USE Options in Effect

When you issue a FOCUS command to access a FOCUS database, such as TABLE FILE *filename*, FOCUS searches for a Master File with the specified file name, and then searches for a database with the same file name in CMS or allocated to the same ddname in MVS.

- In CMS FOCUS, the FOCUS file has the default file type FOCUS and the default file mode A.

For example, the command TABLE FILE EMPLOYEE uses the file EMPLOYEE FOCUS A for CMS or the file allocated to ddname EMPLOYEE for MVS.

If the FOCUS file has a fileid different from these defaults, you must issue the USE command to identify the file, with its file specifications, and associate it with a specific Master File.

The USE command specifies the names and locations of FOCUS databases for the following conditions:

- Default naming conventions are not used.
- You need an additional option. For example, you can protect databases from change or concatenate several similar databases.

When you identify FOCUS databases with the USE command, a USE directory is created, which is a list of database definitions. When a USE directory is in effect, FOCUS will locate databases using the information in the directory, instead of searching for the file using default names. A USE directory enables you to access up to 255 databases. The USE directory applies only to FOCUS databases.

## *Syntax* USE Command Syntax

The syntax of the USE command is

```
USE action
fileid [READ|NEW] [AS masternamе]
```

or

```
fileid AS mastername ON server READ
```

or

```
fileid LOCAL
```

or

```
fileid ON server
.
.
.
ind {WITH|INDEX} mastername
END
```

where:

*action*

Is one of the following:

`ADD` appends one or more new fileids to the present directory. If you issue the USE command without the ADD parameter, the list of databases you specify replaces the existing USE directory.

`CLEAR` erases the USE directory. The keyword END is not required with this option. Any other options specified will be ignored.

`REPLACE` replaces an existing fileid in the USE directory. This option enables you to change the file specification for the fileid and the options following the fileid.

*fileid*

Is any valid file name for the specific operating system.

| For this platform... | The fileid is... |
|---|---|
| For CMS | Any valid file name, in *filename filetype filemode* format containing your FOCUS file. |
| For MVS | A *ddname* allocated to the MVS data set containing your FOCUS file. |

`READ`
: Restricts databases to read-only access.

`NEW`
: Indicates that the database has yet to be created.

`AS` *`mastername`*
: Specifies the name of the Master File to be associated with the fileid.

`ON` *`server`*
: Specifies the userid of the FOCUS Database Server (sink machine) that synchronizes FOCUS databases for use by multiple users on CMS, or the communications data set of the FOCUS Database Server on MVS.

`LOCAL`
: This option requires a previous directory entry for the fileid with the ON *server* option. For CMS, accesses an SU database directly through the operating system. Before using this option you must link and access the minidisk on which the SU database resides in read-only mode.

  For MVS, accesses an SU database using the Multi-Threaded SU Reporting Facility. Before using this option you must allocate the SU database in SHR mode.

*`ind`*
: Is the fileid (on CMS) or ddname (on MVS) of an external index.

`WITH|INDEX`
: Establishes the relationship between an external index and the component database. INDEX is a synonym for WITH.

The following options after the fileid are valid together:

```
READ and AS
NEW and AS
AS and ON and READ
```

Any other combination of options after the fileid is not valid.

## *Syntax* How to Specify Multiple Databases

You can specify several databases in one USE command, each with different parameters. For example:

```
USE
fileid1 ON MULTID
fileid2 AS PRODUCTS
fileid3 READ AS ACCOUNTS
END
```

*Syntax* **How to Erase the USE Directory**

To erase the USE directory, enter the following command:

```
USE CLEAR
```

# Using Alternative File Specifications

If the FOCUS database has a fileid other than the default, issue the USE command to identify the file, with its file specifications, and associate it with a specific Master File.

*Syntax* **How to Specify Alternative File Specifications**

The syntax is

```
USE
fileid AS mastername
END
```

where:

*fileid*
Is any valid file specification for the specific operating system.

*mastername*
Is name of the Master File name that will be associated with the fileid.

*Example* **Specifying Different File Names**

To read the database with the name EMP026 described by the Master File EMPLOYEE, enter this USE command:

For CMS:

```
USE
EMP026 FOCUS A AS EMPLOYEE
END
```

For MVS:

```
USE
EMP026 AS EMPLOYEE
END
```

After entering the USE command, you can read the EMP026 database by entering the command TABLE FILE EMPLOYEE.

### *Example* Specifying Different File Types and Extensions

For CMS: To read the database with the name EMP026 and a file type of FOCUS on the A-disk, described by the Master File EMPLOYEE, enter this USE command:

```
USE
EMP026 FOCUS A AS EMPLOYEE
END
```

After entering the USE command, you can read the EMP026 database by entering the command TABLE FILE EMPLOYEE.

### *Example* Specifying Different File Locations

For CMS: To read the database with the name EMP026 located on the F disk, described by the Master File EMPLOYEE, enter this USE command:

```
USE
EMP026 FOCUS F AS EMPLOYEE
END
```

The first database in the USE directory defines the default file type and file mode for the rest of the session or until you clear the USE directory. For example, if you later issue the command TABLE FILE PRODUCT, FOCUS searches for the database PRODUCT FOCUS F even if you did not specify the database in the USE command. If you want to read both EMPLOYEE FOCUS F and PRODUCT FOCUS A, issue:

```
USE
EMP026 FOCUS F AS EMPLOYEE
PRODUCT FOCUS A
END
```

Since PRODUCT FOCUS A is the second entry in the USE directory, the default file mode remains F.

# Identifying New Databases to FOCUS

The parameter NEW in the USE command identifies databases that do not exist yet. When you identify a new database, you can accept the default file specification conventions or specify different ones.

In CMS, when you issue a MODIFY command specifying a database that does not exist, FOCUS creates the database with a default file name, file type and file mode. You can issue the USE command with the NEW parameter to give the database a fileid other than the default.

For MVS, you must allocate the file, with the MVS command ALLOCATE or the FOCUS command DYNAM, before you issue the USE command.

The syntax is

```
USE
fileid  NEW
END
CREATE file mastername
```

where:

*fileid*
: Is any valid file specification for the operating system. The fileid will be assigned to the database later in the session when the actual create happens.

*mastername*
: Is the name of the Master File associated with the database.

If you omit the NEW parameter, a message is returned stating that the database cannot be found, and the USE command is not executed.

***Example*** **Identifying a New Database**

To create the database WAGES using the WAGES Master File, enter the following:

For CMS:

```
USE
WAGES FOCUS F NEW
END
CREATE FILE WAGES
```

For MVS:

```
USE
WAGES NEW
END
CREATE FILE WAGES
```

# Protecting Files (Read Only)

You can protect databases from changes by issuing USE commands with the READ parameter. Protected databases can be read by various FOCUS tools and commands such as MODIFY and SCAN, but cannot be changed. The syntax is

```
USE
fileid READ
END
```

where:

*fileid*
Is any valid file specification for the operating system.

***Example*** **Protecting a Database**

For example, to protect the database EMPLOYEE, enter:

For CMS:

```
USE
EMPLOYEE FOCUS A READ
END
```

For MVS:

```
USE
EMPLOYEE READ
END
```

# Concatenating Databases

If several FOCUS databases are described by the same Master File, you can read all of the databases in one TABLE or GRAPH request by issuing a USE command that concatenates all of the databases. The syntax is as follows

```
USE
fileid1 AS masternamе
fileid2 AS masternamе
.
.
fileidn AS masternamе
END
```

where:

*fileid1...*
Are any valid file specifications, for the operating system, for the files being concatenated.

*mastername*
Is the name of the Master File that describes the databases.

### *Example* Concatenating Databases to One Master File

For example, to read three FOCUS databases: EMP024, EMP025, and EMP026, all described by the Master File EMPLOYEE, issue the following USE command:

For CMS:

```
USE
EMP024 FOCUS A AS EMPLOYEE
EMP025 FOCUS C AS EMPLOYEE
EMP026 FOCUS C AS EMPLOYEE
END
```

For MVS:

```
USE
EMP024 AS EMPLOYEE
EMP025 AS EMPLOYEE
EMP026 AS EMPLOYEE
END
```

You can then read all three databases with the single command TABLE FILE EMPLOYEE.

### *Example* Concatenating Multiple Master Files

You can concatenate databases to several Master Files in one USE command. For example, the following USE command concatenates the EMP01 and EMP02 databases to the Master File EMPLOYEE, and concatenates the SALES01 and SALES02 databases to the Master File SALES:

For CMS:

```
USE
EMP01 FOCUS A AS EMPLOYEE
EMP02 FOCUS A AS EMPLOYEE
SALES01 FOCUS A AS SALES
SALES02 FOCUS A AS SALES
END
```

For MVS:

```
USE
EMP01 AS EMPLOYEE
EMP02 AS EMPLOYEE
SALES01 AS SALES
SALES02 AS SALES
END
```

To read the EMP01 and EMP02 databases, begin by entering

```
TABLE FILE EMPLOYEE
```

and to read the SALES01 and SALES02 databases, begin by entering:

```
TABLE FILE SALES
```

### *Example* Concatenating Multiple Databases and a Single Cross-Reference File

To read multiple files having a cross-reference file as one file, specify the host files in the USE command and then the cross-reference file.

For example, the file EMPLOYEE is made up of two files EMP01 and EMP02 that reference a common cross-reference file EDUCFILE. To read the two files together, enter the following USE command:

For CMS:

```
USE
EMP01 FOCUS A AS EMPLOYEE
EMP02 FOCUS A AS EMPLOYEE
EDUCFILE FOCUS A
END
```

For MVS:

```
USE
EMP01 AS EMPLOYEE
EMP02 AS EMPLOYEE
EDUCFILE
END
```

### *Example* Concatenating Multiple Databases and Multiple Cross-Reference Files

If the EMPLOYEE database consisted of two files, EMP01 and EMP02, and each had its own cross-reference file, ED01 and ED02, you can read all four files in one command by entering this USE command where each host file is followed by its cross-reference.

You cannot specify a concatenated file as the cross-referenced file in a JOIN command.

You can take an indexed view of a concatenated file by creating an external index database and using the TABLE FILE *filename.indexed_fieldname* command. For more information about indexed views, see the instructions for creating an external index database in the *Maintaining Databases* manual.

For CMS:

```
USE
EMP01 FOCUS A AS EMPLOYEE
ED01  FOCUS A AS EDUCFILE
EMP02 FOCUS A AS EMPLOYEE
ED02  FOCUS A AS EDUCFILE
END
```

For MVS:

```
USE
EMP01 AS EMPLOYEE
ED01  AS EDUCFILE
EMP02 AS EMPLOYEE
ED02  AS EDUCFILE
END
```

# Specifying Databases in Simultaneous Usage Mode

In CMS, the FOCUS Database Server is a disconnected virtual machine which manages all READ/WRITE operations to a FOCUS database.

In MVS, the FOCUS Database Server is a batch job or started task managing all READ/WRITE operations to a FOCUS database.

```
USE
fileid ON server
END
```

where:

*fileid*

In CMS, is the file name, file type, and file mode of the FOCUS database accessed by the disconnected virtual machine (FOCUS Database Server).

In MVS, is the ddname of the FOCUS database allocated in the batch job or started task.

### *Example* Accessing an SU Database in CMS

If you want to use the EMPLOYEE FOCUS file on the A-disk of the FOCUS Database Server named myserver, code the following:

```
USE
EMPLOYEE FOCUS A ON MYSERVER
END
```

### *Example* Accessing an SU Database in MVS

If you want to use the EMPLOYEE FOCUS file allocated to the database server batch job or started task, two things must be done:

1. You must allocate a ddname to the communications data set that is allocated in the FOCUS Database Server batch job or started task pointing to the ddname FOCSU.

   For example,

   ```
   DYNAM ALLOC FILE MYSERVER DS prefix.FOCSU.DATA SHR
   ```

2. You must issue the USE command for your database allocated in the batch job or started task.

   ```
   USE
   EMPLOYEE ON MYSERVER
   END
   ```

## Multi-Thread Configuration

Performance gains may be achieved by routing READ only requests directly to the file on disk instead of going through the FOCUS Database Server. This is called a multi-thread configuration. It is accomplished with the USE command and the keyword LOCAL.

In CMS, first link and access the database in READ only mode and issue the USE LOCAL syntax:

```
CMS CP LINK MYSERVER 191 391 RR
CMS ACCESS 391 B
USE
EMPLOYEE FOCUS A ON MYSERVER
EMPLOYEE FOCUS B LOCAL
END
```

In MVS, allocate the FOCUS database and issue the USE LOCAL syntax:

```
DYNAM ALLOC FILE EMPLOYEE DS prefix.EMPLOYEE.FOCUS SHR
DYNAM ALLOC FILE MYSERVER DS prefix.FOCSU.DATA SHR
USE
EMPLOYEE ON MYSERVER
EMPLOYEE LOCAL
END
```

For more information about Simultaneous Usage Mode, see the *FOCUS Simultaneous Usage Reference Manual for CMS* or the FOCUS *Simultaneous Usage Reference Manual for MVS.*

**Note:**

- On a FOCUS Database Server, 255 files can be open at one time with 256 users connected.
- The READ option is available for accessing an SU database with an alternate Master File (using an AS name). For example:

  ```
  USE EMP01 AS EMPLOYEE ON MULTID READ
  ```

  In an SU environment, the READ option does not provide Read-only access.

# Using the LOCATION Attribute

The file type and physical location of a file that is named by the LOCATION attribute in the Master File, defaults to FOCUS in the local directory unless a USE command is issued. If the physical files are on different disks or have different file types, they must be listed in the USE list.

# Displaying the USE Options in Effect

To display USE options in effect, enter the ? USE query in a stored procedure:

```
? USE
```

This query displays a list of databases you specified with the USE commands, with options currently in effect.

***Example*** **Displaying USE Options**

A sample output from the ? USE command is:

```
? USE
 DIRECTORIES IN USE ARE:
 CAR
 EMPLOYEE
 JOBFILE
 EDUCFILE
```

# 11 Providing File Security: DBA

**Topics:**

- Introduction
- Database Security
- Special Considerations
- Limiting the Access: The RESTRICT Attribute
- Placing Security Information in a Central File: The DBAFILE Attribute
- FOCEXEC Security
- Program Accounting/Resource Limitation
- Absolute File Integrity

As Database Administrator, you can use FOCUS security features to provide security for any FOCUS database. You can use FOCUS security features to limit the number of records or reads a user can request in a report. You can also use user-written programs to perform program accounting on FOCUS files. You can use the Usage Accounting and Security Exit Routine (UACCT) to collect usage statistics and data on attempted access violations.

You can also use FOCUS security features to provide security for non-FOCUS files. However, the RESTRICT command (*Restricting Existing Files* on page 11-29) is not available. Note that FOCUS security cannot protect a file from non-FOCUS access.

# Introduction

FOCUS provides a number of security options:

- You can limit the users who have access to a given file using the USER attribute discussed in *Identifying Users: The USER Attribute* on page 11-8.
- You can restrict a user's access rights to read, write, or update only using the ACCESS attribute discussed in *Specifying the Type of Access: The ACCESS Attribute* on page 11-10.
- You can restrict a user's access to certain fields or segments using the RESTRICT attribute discussed in *Limiting the Access: The RESTRICT Attribute* on page 11-15.
- You can ensure that only records that pass a validation test are retrieved using the RESTRICT attribute discussed in *Limiting the Access: The RESTRICT Attribute* on page 11-15.
- You can limit the values a user can write to the database or you can limit which values a user can alter using the RESTRICT attribute discussed in *Limiting the Access: The RESTRICT Attribute* on page 11-15.
- You can point to passwords and restrictions stored in another Master File with the DBAFILE attribute discussed in *Placing Security Information in a Central File: The DBAFILE Attribute* on page 11-22.
- You can use the FOCUSID exit routine to let an external security system set the FOCUS password.
- You can place security on FOCEXECs, which is discussed in *FOCEXEC Security* on page 11-31.

Program accounting, resource limitation, and the UACCT exit routine are discussed in *Program Accounting/Resource Limitation* on page 11-34.

Also, whenever a new file is created, you have to decide whether or not to invoke the FOCUS shadow paging feature, which guarantees the integrity of the data in the file. The shadow paging feature is discussed in *Absolute File Integrity* on page 11-37.

# Database Security

Implementing FOCUS security features is a straightforward process in which you specify:

- The names or passwords of FOCUS users granted access to a file.
- The type of access the user is granted.
- The segments, fields, or ranges of data values to which the user's access is restricted.

You provide FOCUS security on a file-by-file basis.

The following is a Master File of a file that uses FOCUS security features:

```
FILENAME = PERS, SUFFIX = FOC,$
SEGMENT = IDSEG, SEGTYPE = S1,$
 FIELD = SSN          ,ALIAS = SSN   ,FORMAT = A9   ,$
 FIELD = FULLNAME     ,ALIAS = FNAME ,FORMAT = A40  ,$
 FIELD = DIVISION     ,ALIAS = DIV   ,FORMAT = A8   ,$
SEGMENT=COMPSEG, PARENT=IDSEG, SEGTYPE=S1,$
 FIELD = SALARY       ,ALIAS = SAL   ,FORMAT = D8   ,$
 FIELD = DATE         ,ALIAS = DATE  ,FORMAT = YMD  ,$
 FIELD = INCREASE     ,ALIAS = INC   ,FORMAT = D6   ,$
END
DBA=JONES76,$
USER=TOM   ,ACCESS=RW, $
USER=BILL  ,ACCESS=R  ,RESTRICT=SEGMENT      ,NAME=COMPSEG  ,$
USER=JOHN  ,ACCESS=R  ,RESTRICT=FIELD        ,NAME=SALARY   ,$
                                              NAME=INCREASE ,$
USER=LARRY ,ACCESS=U  ,RESTRICT=FIELD        ,NAME=SALARY   ,$
USER=TONY  ,ACCESS=R  ,RESTRICT=VALUE        ,NAME=IDSEG,
   VALUE=DIVISION EQ 'WEST' ,$
USER=MARY  ,ACCESS=W  ,RESTRICT=VALUE        ,NAME=SALTEST,
   VALUE=INCREASE+SALARY GE SALARY,$
                                              NAME=HISTTEST,
   VALUE=DIV NE ' ' AND DATE GT 0,$
```

The declarations (called security declarations) including and following the word END tell FOCUS that security is needed for the file and what type of security you want. Each security declaration can consist of one or several of the following attributes:

- The DBA attribute gives the name or password of the Database Administrator for the file. The Database Administrator has unlimited access to the file and its Master File.
- The USER attribute identifies a user as a legitimate user of the file. Only users whose name or password is specified in the Master File of a FOCUS file with security placed on it have access to that file.
- The ACCESS attribute defines the type of access to the file that a given user has. The four types of access available are:

  RW, which allows a user to both read and write to a file.

  R, which allows a user to read data in a file only.

  W, which allows a user to write new segment instances to a file only.

  U, which allows a user to update records in a file only.
- The RESTRICT attribute specifies certain segments or fields to which the user is not granted access. It can also be used to restrict the data values a user can see or perform transactions on.
- The NAME and VALUE attributes are part of the RESTRICT declaration.

You describe your file security by specifying various values for these attributes in a comma-delimited format, just as you specify any other attribute in the Master File.

The word END on a line by itself in the Master File terminates the segment and field attributes and indicates that the access limits follow. If you place the word END in a Master File, it must be followed by at least a DBA attribute.

## *Reference* Special Considerations

When using the JOIN command it is possible to bypass the DBA information in a FOCUS file. This is a security exposure created because in a JOIN structure the DBA information is read from the host Master File. This problem is solved by using the DBAFILE feature discussed in *Placing Security Information in a Central File: The DBAFILE Attribute* on page 11-22. All files in the joined structure will get security information as coded in the DBAFILE.

# Identifying the DBA: The DBA Attribute

The first line of the access limits should be an arbitrary code name that identifies the Database Administrator. This password can be up to eight characters long. Since nothing else is needed, this line is terminated by the usual delimiter (,$). For example:

```
DBA=JONES76,$
```

**Note:**

- Every file having access limits must have a DBA.
- Groups of cross-referenced files must have the same DBA value.
- Partitioned files which are read together in the USE command must have the same DBA value.
- The Database Administrator has unlimited access to the file and all cross-referenced files. Therefore, no field, segment, or value restrictions can be specified with the DBA attribute.
- You cannot encrypt and decrypt Master Files or restrict existing data files without the DBA password.

The DBA has the freedom to change any of the security attributes. If you change the DBA password in the Master File, you must use the RESTRICT command for existing files at the FOCUS command level (discussed in *Restricting Existing Files* on page 11-29) to inform each FOCUS file affected by the change. Unless this is done, FOCUS will assume that the new description is an attempt to bypass the restriction rules. You use the following procedure for each file affected:

1. SET PASS=*old*_DBA_*value*
2. Issue the command:
   RESTRICT
   *filename*
   END
3. SET PASS=*new_value*

**Note:** See the *Overview and Operating Environments* manual for specific syntax of the RESTRICT command.

You should thoroughly test every security attribute before the file is used. It is particularly important to test the VALUE limits to make sure they do not contain errors. Value tests are executed as if they were extra screening conditions or VALIDATE statements typed after each request statement. Since users are unaware of the value limits, errors caused by the value limits may confuse them.

## Including the DBA Attribute in HOLD Files

With the SET HOLDSTAT command (described in the SET chapter in the *Developing Applications* manual), you can identify a file containing DBA information and comments to be automatically included in HOLD and PCHOLD Master Files.

For MVS, the file must be a member in the PDS allocated to ddname MASTER or ERRORS (see the *Overview and Operating Environments* manual for more information); for CMS, it must have file type MASTER or ERRORS (see the *Overview and Operating Environments* manual for more information). In both cases, MASTER takes precedence over ERRORS.

The Information Builders-supplied file is named HOLDSTAT; user-specified HOLDSTAT files can have any valid file name.

The HOLDSTAT file must contain a dollar sign ($) in column 1. The keyword $BOTTOM in the file indicates there is DBA information to be added.

The following sample HOLDSTAT is included with FOCUS Version 7.0:

```
$==============================================================$
$     HOLD file created on &DATE at &TOD by FOCUS &FOCREL       $
$             Database records retrieved= &RECORDS              $
$             Records in the HOLD file = &LINES                 $
$==============================================================$
```

To include DBA information in HOLD Master Files, use the following syntax at the bottom of the HOLDSTAT file:

```
$BOTTOM
END
DBA=...
```

**Note:** User-defined variables may not be included in the comments portion of the HOLDSTAT file. Other DBA attributes can be included in the HOLDSTAT file as can system variables.

All lines from the HOLDSTAT file that appear prior to $BOTTOM are placed at the top of the HOLD Master File, before any file and field declarations. All lines that appear after $BOTTOM are appended to the bottom of the HOLD Master File. Any Dialogue Manager variables are replaced with the actual variable values. The following example illustrates the use of HOLDSTAT. The TABLE request is:

```
SET HOLDSTAT = ON
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME SALARY
BY EID
ON TABLE HOLD
END
```

It produces the HOLD Master File:

```
$================================================================$
$    HOLD file created on 1999/05/20 at 17.58.10 by FOCUS 7.0    $
$            Database records retrieved=       19                $
$            Records in the HOLD file =        19                $
$================================================================$
FILE = HOLD   ,SUFFIX = FIX
SEGNAME = HOLD, SEGTYPE = S01
FIELDNAME = EMP_ID             ,E01         ,A9       ,A12      ,$
FIELDNAME = LAST_NAME          ,E02         ,A15      ,A16      ,$
FIELDNAME = FIRST_NAME         ,E03         ,A10      ,A12      ,$
FIELDNAME = SALARY             ,E04         ,D12.2M   ,D08      ,$
```

The next example illustrates the use of a user-specified file containing DBA information. The HOLD Master File that is generated contains DBA information from the file name specified in the SET HOLDSTAT command. The HOLDDBA Master File is:

```
$================================================================$
$     HOLD file created on &DATE at &TOD by FOCUS &FOCREL       $
$             Database records retrieved= &RECORDS              $
$             Records in the HOLD file = &LINES                 $
$================================================================$
$BOTTOM
END
DBA=MARY,$
```

The following TABLE request uses the HOLDDBA Master File:

```
SET HOLDSTAT = HOLDDBA
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME SALARY
BY EID
ON TABLE HOLD
END
```

The HOLD Master File that results is:

```
$================================================================$
$    HOLD file created on 1999/05/20 at 17.58.10 by FOCUS 7.0   $
$             Database records retrieved=       19               $
$             Records in the HOLD file =        19               $
$================================================================$
FILE = HOLD ,SUFFIX = FIX
SEGNAME = HOLD, SEGTYPE = S01
FIELDNAME = EMP_ID            ,E01         ,A9       ,A12     ,$
FIELDNAME = LAST_NAME         ,E02         ,A15      ,A16     ,$
FIELDNAME = FIRST_NAME        ,E03         ,A10      ,A12     ,$
FIELDNAME = SALARY            ,E04         ,D12.2M   ,D08     ,$
END
DBA=MARY,$
```

# Identifying Users: The USER Attribute

The USER attribute is an arbitrary code name that identifies the users who have legitimate access to the file. A USER attribute cannot be specified alone; it must be followed by at least an ACCESS restriction (discussed in *Specifying the Type of Access: The ACCESS Attribute* on page 11-10) to specify what sort of ACCESS the user is granted.

Before using a secured file, a user must enter his or her password using the SET PASS command. If that password is not included in the Master File, the user is denied access to the file. When the user does not have a password or has one that is inadequate for the type of access requested, the following message is displayed:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE: filename
```

## Setting the USER Attribute

Any user whose name or password is not declared in the Master File is denied access to that file. The syntax of the USER attribute is

```
USER = name,
```

where:

*name*
: Is an arbitrary code name of up to eight characters for the user.

For example:

```
USER=TOM,...
```

You can specify a blank password. Such a password does not require the user to issue a SET PASS= command. A blank password may still have access limits and is convenient when a number of users have the same access rights. An example of setting a user's password to blank, and access to read only follows:

```
USER= , ACCESS=R,$
```

## Establishing User Identity

A user must enter his or her password before using any FOCUS file that has security specified for it. A single user may have different passwords in different files. For example, in file ONE the rights of password BILL apply, but in file TWO the rights of password LARRY apply. Use the SET PASS command to establish the passwords. The syntax is

where:

```
SET {PASS|USER} = name [ [IN {file|* [NOCLEAR]} ] , name [IN file] ... ]
```

*name*
: Is the user's name or password.

*
: Indicates that name replaces all passwords active in all files.

NOCLEAR
: Provides a way to replace all passwords in the list of active passwords while retaining the list.

In the following example, the password TOM is in effect for all files that do not have a specific password designated for them:

```
SET PASS=TOM
```

For the next example, in file ONE the password is BILL, and in file TWO the password is LARRY. No other files have passwords set for them:

```
SET PASS=BILL IN ONE, LARRY IN TWO
```

Here, all files have password SALLY except files SIX and SEVEN, which have password DAVE:

```
SET PASS=SALLY, DAVE IN SIX
SET PASS=DAVE IN SEVEN
```

The password is MARY in file FIVE and FRANK in all other files:

```
SET PASS=MARY IN FIVE,FRANK
```

FOCUS maintains a list of the files for which a user has set specific passwords. To see the list of files, issue:

```
? PASS
```

When the user sets a password IN * (all files), the list of active passwords collapses to one entry with no associated file name. To retain the file name list, use the NOCLEAR option.

In the next example, the password KEN replaces all passwords active in all files, and the table of active passwords is folded to one entry:

```
SET PASS=KEN IN *
```

In the following, MARY replaces all passwords in the existing table of active passwords (which consists of files NINE and TEN) but FRANK is the password for all other files. The option NOCLEAR provides a shorthand way to replace all passwords in a specific list:

```
SET PASS=BILL IN NINE,TOM IN TEN
SET PASS=MARY IN * NOCLEAR,FRANK
```

**Note:** The FIND function does not work with COMBINEd files secured with different passwords.

Users must issue their passwords using the SET PASS command during each FOCUS session in which they use a secured file. They may issue their passwords at any time before using the file and can issue a different password afterward to access another file.

## Specifying the Type of Access: The ACCESS Attribute

The ACCESS attribute specifies what sort of access to the file a user is granted. Every security declaration, except the DBA, must have a USER attribute and an ACCESS attribute.

The following is a complete security declaration, consisting of a USER attribute and an ACCESS attribute.

```
USER=TOM, ACCESS=RW,$
```

This declaration gives Tom read and write (for adding new segment instances) access to the file.

You can assign the ACCESS attribute one of four values. These are:

| | |
|---|---|
| `ACCESS=R` | Read only |
| `ACCESS=W` | Write only |
| `ACCESS=RW` | Read the file and write new segment instances |
| `ACCESS=U` | Update only |

Access levels affect what kind of FOCUS commands a user can issue. Before you decide what access levels to assign to a user, you must consider what commands that user will need. If a user does not have sufficient access rights to use a given command, the following error message will be displayed:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE: filename
```

ACCESS levels determine what a user can do to the file. You use the RESTRICT attribute (discussed in *Limiting the Access: The RESTRICT Attribute* on page 11-15) to limit the fields, values, or segments to which a user has access. Every USER attribute must be assigned an ACCESS attribute. The RESTRICT attribute is optional; without it, the user has unlimited access to fields and segments within the file.

The type of access granting use of various FOCUS commands is shown in the following table. When more than one type of access is shown, any type of access marked will allow the user at least some use of that command. Often, however, the user will be able to use the command in different ways, depending on the type of access he or she is granted.

**Types of Access**

| Command | R | W | RW | U | DBA |
|---|---|---|---|---|---|
| **CHECK** | X | X | X | X | X |
| **CREATE** | | | X | | X |
| **DECRYPT** | | | | | X |
| **DEFINE** | X | | X | | X |
| **ENCRYPT** | | | | | X |
| **FSCAN** | | X | X | X | X |
| **HLI** | | | X | | X |
| **MODIFY** | | X | X | X | X |
| **REBUILD** | | | X | | X |
| **RESTRICT** | | | | | X |
| **SCAN** | | | X | X | X |
| **TABLE** | X | | X | | X |

## The CHECK Command

Users without the DBA password or read/write access are allowed limited access to the CHECK command. However, when the HOLD option is specified, the warning ACCESS LIMITED BY PASSWORD is produced, and restricted fields are propagated to the HOLD file depending on the DBA RESTRICT attribute. Refer to *Limiting the Access: The RESTRICT Attribute* on page 11-15 for more information on the RESTRICT attribute.

The RESTRICT attribute keywords affect the resulting HOLD file as follows:

| | |
|---|---|
| `FIELD` | Fields named with the NAME parameter are not included in the HOLD file. |
| `SEGMENT` | The segments named with the NAME parameter are included, but fields in those segments are not. |
| `SAME` | The behavior is the same as for the user named in the NAME parameter. |
| `NOPRINT` | Fields named in the NAME or SEGNAME parameter are included since these can be referenced by the user. |
| `VALUE` | Fields named in the VALUE parameter are included since these can be referenced by the user. |

**Note:** RESTRICT=PROGRAM has no effect on CHECK FILE HOLD.

If you issue the CHECK command with the PICTURE option, the RESTRICT attribute keywords affect the resulting picture as follows:

| | |
|---|---|
| `FIELD` | Fields named with the NAME parameter are not included in the picture. |
| `SEGMENT` | The boxes appear for segments named with the NAME parameter, but fields in those segments do not. |
| `SAME` | The behavior is the same as for the user named in the NAME parameter. |
| `NOPRINT` | This option has no effect on the picture. |
| `VALUE` | This option has no effect on the picture. |

## The CREATE Command

Only users with the DBA password or read/write (RW) access rights can issue a CREATE command.

## The DECRYPT Command

Only users with the DBA password can issue a DECRYPT command.

## The DEFINE Command

As with all reporting commands, a user need only have an access of R (read only) to use the DEFINE command. An access of R permits the user to read records from the file and prepare reports from them. The only users who cannot use the DEFINE command are those whose access is W (write only) or U (update only).

## The ENCRYPT Command

Only users with the DBA password can use the ENCRYPT command.

## Host Language Interface (HLI)

In order to have use of the Host Language Interface, a user must have read/write (RW) access. With ACCESS=RW, FIELD and SEGMENT restrictions are active, but VALUE restrictions are not. (See *Limiting the Access: The RESTRICT Attribute* on page 11-15 for information on these restrictions.)

The password is placed in the File Control Block (FCB), words 19 and 29 (byte 73 to 80).

## The MODIFY Command

Users with ACCESS=W, RW, or U can use the MODIFY command. In MODIFY, access of U does not allow the user to use the INCLUDE and DELETE actions; only UPDATE operations are permitted. Both ACCESS=RW and ACCESS=W allow full use of all the MODIFY features. New instances of data may be added to a file and old ones deleted; existing values may be updated. Users with ACCESS=R cannot use the MODIFY command.

## The REBUILD Command

Only users with the DBA password or read/write (RW) access rights can issue the REBUILD command.

## The RESTRICT Command

Only users with the DBA password may use the RESTRICT command.

## The FSCAN Facility

Users with ACCESS=RW have unlimited access to the file, except for any restrictions imposed by the RESTRICT or NAME attributes. Users with ACCESS=U can display the entire file, except for any restrictions imposed by the RESTRICT or NAME attributes; however, users with ACCESS=U cannot input or delete instances and can update non-key fields only. Users whose access to any portion of the file is limited to ACCESS=R cannot use FSCAN.

FSCAN honors DBA security restrictions on segments and fields; it prohibits display of those segments and fields from which the user is restricted. FSCAN does not honor DBA field value restrictions and will display all field values regardless of the user.

If the user has no access to a key field in the root segment, that user is blocked from using FSCAN on the database. If the user has no access to a segment, that segment is not listed on the menu that appears when the user enters the CHILD command.

## The SCAN Facility

The rules for accessing a file are the same as for FSCAN except that, in addition, users with ACCESS=W cannot use SCAN.

## The TABLE Command

A user who has access of R or RW may use the TABLE command. Users with access of W or U may not.

# Limiting the Access: The RESTRICT Attribute

As we have seen, the ACCESS attribute tells FOCUS what a user can do with a file. The optional RESTRICT attribute further restricts a user's access to certain fields, values, or segments. The syntax of the RESTRICT attribute is as follows

```
...RESTRICT=level, NAME={name|SYSTEM} [, VALUE=test],$
```

where:

*level*

Can be one of the following:

`FIELD` specifies that the fields named with the NAME parameter cannot be accessed by the user.

`SEGMENT` specifies that the segments named with the NAME parameter cannot be accessed by the user.

`PROGRAM` specifies that the program named with the NAME parameter will be called whenever the user uses the file (discussed in *Program Accounting/Resource Limitation* on page 11-34).

`SAME` specifies that the user has the same restrictions as the user named in the NAME parameter. No more than four nested SAME users are valid.

`NOPRINT` specifies that the field named in the NAME or SEGMENT parameter can be mentioned in a request statement, but will not be displayed.

*name*

Is the name of the field or segment you wish to restrict. When used after NOPRINT, this can only be a field name. NAME=SYSTEM, which can only be used with value tests, restricts every segment in the file, including descendant segments. Multiple fields or segments can be specified by issuing the RESTRICT attribute several times for one user.

`VALUE`

Specifies that the user can have access to only those values that meet the test described in the "test" parameter.

*test*

Is the value test that the data must meet before the user can have access to it.

For example:

```
USER=BILL ,ACCESS=R ,RESTRICT=SEGMENT ,NAME=COMPSEG,$
```

**Note:** For write access, if *name* is a segment name, a MATCH *key* ON MATCH/NOMATCH is performed. For any other name, a validate is done without a MATCH.

## Restricting Fields and Segments

The RESTRICT attribute identifies the segments or fields that the user will not be able to access. Anything not named in the RESTRICT attribute will be accessible.

Without the RESTRICT attribute, the user has access to the entire file. Users may be limited to reading, writing, or updating new records, but every record in the file is available for the operation.

The syntax to restrict access to a field or segment is

```
...RESTRICT=level,  NAME=name,$
```

where:

*level*

Can be one of the following:

`FIELD` specifies that the fields named with the NAME parameter cannot be accessed by the user.

`SEGMENT` specifies that the segments named with the NAME parameter cannot be accessed by the user.

`SAME` specifies that the user has the same restrictions as the user named in the NAME parameter.

`NOPRINT` specifies that the field named in the NAME or SEGMENT parameter can be mentioned in a request statement but will not be displayed. When used after NOPRINT, NAME can only be a field name.

In the following example, Bill has read-only access to everything in the file except the COMPSEG segment:

```
USER=BILL ,ACCESS=R ,RESTRICT=SEGMENT ,NAME=COMPSEG,$
```

**Note:**

- If a field or segment is mentioned in the NAME attribute, it cannot be retrieved by the user. If such a field or segment is mentioned in a request statement, it will be rejected as beyond the user's access rights. With NOPRINT, the field or segment can be mentioned, but the data will not be displayed. The data will appear as blanks for alphanumeric format or zeros for numeric fields.
- You can restrict multiple fields or segments by providing multiple RESTRICT statements. For example, if you wish to restrict Harry from using both field A and segment B, you issue the following access limits:

```
USER=HARRY, ACCESS=R,      RESTRICT=FIELD , NAME=A,$
                           RESTRICT=SEGMENT, NAME=B,$
```

- You can restrict as many segments and fields as you like.
- Using RESTRICT=SAME is a convenient way to reuse a common set of restrictions for more than one password. If you specify RESTRICT=SAME and provide a user name or password as it is specified in the USER attribute for the NAME value, the new user will be subject to the same restrictions as the one named in the NAME attribute. You can then add additional restrictions as they are needed. In the following example, both Sally and Harry have the same access privileges as BILL. In addition, Sally is not allowed to read the SALARY field.

```
USER=BILL ,ACCESS=R ,RESTRICT=VALUE ,NAME=IDSEG,
     VALUE=DIVISION EQ 'WEST', $

USER=SALLY ,ACCESS=R ,RESTRICT=SAME  ,NAME=BILL,
                      RESTRICT=FIELD ,NAME=SALARY,$

USER=HARRY ,ACCESS=R ,RESTRICT=SAME  ,NAME=BILL, $
```

## Restricting Values

You can also restrict the values to which a user has access by providing a test condition in your RESTRICT statement. The user is restricted to using only those values that satisfy the test condition.

You can restrict values in one of two ways: you can restrict the values the user can read from the file, or you can restrict what the user can write to a file. These restrictions are two separate functions: one does not imply the other. You use the ACCESS attribute to specify whether the values the user reads or the values the user writes are restricted.

You restrict the values a user can read by setting ACCESS=R and RESTRICT=VALUE. This type of restriction prevents the user from seeing any data values other than those that meet the test condition provided in the RESTRICT statement. A RESTRICT statement with ACCESS=R functions as an involuntary IF statement in a report request. Therefore, the syntax for ACCESS=R value restrictions must follow the rules for an IF test in a report request.

You restrict the values a user can write to a file by setting ACCESS=W and RESTRICT=VALUE. This type of restriction, which functions as a VALIDATE statement in MODIFY, limits the actual values a user can enter. Therefore, the syntax for ACCESS=W value restrictions must follow the rules for a VALIDATE statement in MODIFY. You can also use ACCESS=W and RESTRICT=VALUE to limit the data values in the file for which a user can provide new values. When ACCESS=W, the user will be able to access all data values in the file. The user will simply be prohibited from entering certain values or new values for certain existing values.

If you want to prevent a user both from entering certain values and from seeing other values, you must issue two RESTRICT statements: one with ACCESS=W, which limits the values a user can write or alter, and one with ACCESS=R, which limits the values the user can see. ACCESS=RW is meaningless with a RESTRICT=VALUE statement.

**Note:** You can display a table listing users and their access privileges with the EX DBATABLE command described in *Displaying the Decision Table* on page 11-30. For DBATABLE to work properly, you must list all users who have no value restrictions prior to users with value restrictions in the Master File.

## Restricting Values a User Can Read

You restrict the values a user can read with the following syntax

```
...ACCESS=R, RESTRICT=VALUE, NAME=name, VALUE=test,$
```

where:

*name*
: Is the name of the segment on which you are performing the tests. To specify all segments in the file, specify NAME=SYSTEM.

*test*
: Is the test being performed.

For example:

```
USER=TONY ,ACCESS=R ,RESTRICT=VALUE ,NAME=IDSEG,
     VALUE=DIVISION EQ 'WEST' ,$
```

With this restriction, Tony can only see records from the western division.

You type the test expression after VALUE=. The syntax of the test condition is the same as that used by the TABLE command to screen records, except the word IF does not precede the phrase. (Screening conditions in the TABLE command are discussed in the *Creating Reports* manual.) Should several fields have tests performed on them, separate VALUE statements must be provided. Each test must name the segment to which it applies. For example:

```
USER=DICK ,ACCESS=R ,RESTRICT=VALUE ,NAME=IDSEG,
     VALUE=DIVISION EQ 'EAST' OR 'WEST',$
     NAME=IDSEG,
     VALUE=SALARY LE 10000,$
```

If a single test condition exceeds the allowed length of a line, it can be provided in sections. Each section must start with the attribute VALUE= and end with the terminator (,$). For example:

```
USER=SAM, ACCESS=R, RESTRICT=VALUE ,NAME=IDSEG,
     VALUE=DIVISION EQ 'EAST' OR 'WEST',$
     VALUE=OR 'NORTH' OR 'SOUTH',$
```

**Note:** The second and subsequent lines of a value restriction must begin with the keyword OR.

You can apply the test conditions to the parent segments of the data segments on which the tests are applicable. Consider the following example:

```
USER=DICK ,ACCESS=R ,RESTRICT=VALUE ,NAME=IDSEG,
     VALUE=DIVISION EQ 'EAST' OR 'WEST',$
     NAME=IDSEG,
     VALUE=SALARY LE 10000, $
```

The field named SALARY is actually part of a segment named COMPSEG. Since the test is specified with NAME=IDSEG, however, the test is made effective for requests on its parent, IDSEG. In this case, the request PRINT FULLNAME would only print the full names of people who meet this test, that is, whose salary is less than or equal to $10,000, even though the test is performed on a field that is part of a descendant segment of IDSEG. If, however, the test was made effective on COMPSEG, that is, NAME=COMPSEG, then the full name of everyone in the file could be retrieved, but with the salary information of only those meeting the test condition.

## Restricting Values a User Can Write

If a user’s access rights are either W or U, VALUE tests used with the MODIFY command validate new transactions. The format of the test conditions are those used in the ON MATCH VALIDATE expressions of the MODIFY command, which is discussed in the *Maintaining Database* manual.

There are two different ways you can restrict the values a user can write to a file: you can restrict the values the user actually is allowed to enter, or you can restrict the values that the user is allowed to change. You must supply an ACCESS=R restriction to restrict the user from seeing certain data values in the file.

The simplest type of write restriction is one that prevents the user from entering certain values. Thus, it can be used to enforce editing restrictions. For instance, you use this type of restriction to prevent MODIFY users from entering nonsensical values, such as a salary of $10. You can also use this type of restriction to restrict the key values a user is allowed to enter.

The syntax of this type of value test is

```
...ACCESS=W, RESTRICT=VALUE, NAME=name, VALUE=test,$
```

where:

*name*
Is an arbitrary value used as the validate field name.

*test*
Is the test being performed.

This type of value test does not require data file values. Since it does not use the data file, you can supply an arbitrary name for the NAME attribute. The expressions are based entirely on transaction values and can be applied to the transaction immediately after reading it. For example

```
(A) USER=CHUCK ,ACCESS=W ,RESTRICT=VALUE ,
         NAME=CHRANGE,
         VALUE=SALARY LT 20000 AND SALARY GT 5000,$
```

prevents Chuck from entering a salary that is greater than 20,000 or less than 5000. If you use an arbitrary value for NAME=, as shown above, you have created a global restriction similar to the VALIDATE command in MODIFY.

If your MODIFY procedure contains MATCH statements, you may want to restrict the values a user can enter on a segment level by supplying a segment name for NAME=. This creates a condition similar to an ON MATCH VALIDATE phrase. The syntax of this value test is

```
...ACCESS=W, RESTRICT=VALUE, NAME=name, VALUE=test,$
```

where:

*name*
Is the name of the segment on which you perform the test.

*test*
Is the test being performed.

For example:

```
(B) USER=CHUCK ,ACCESS=W ,RESTRICT=VALUE ,NAME=COMPSEG,
         VALUE=SALARY LT 20000 AND SALARY GT 5000,$
```

The difference between the restriction created by example B and that created by example A has to do with your MODIFY procedures. The conditions in the global restriction created by example A are applied prior to MATCH logic in the MODIFY request. The conditions created by example B are applied after your first ON MATCH condition right before the action (UPDATE or DELETE) and can reference D. fields.

You can also restrict the values a user with ACCESS=W can alter. This type of restriction is dependent on the values that are currently in the file and prevents the user from changing certain records. The user will be allowed to perform actions only on the records that pass the validation test.

The syntax of this type of value test is

```
...ACCESS=W, RESTRICT=VALUE, NAME=name, VALUE=test,$
```

where:

*name*
: Is the name of the segment on which you perform the test.

*test*
: Is the test being performed.

For example:

```
USER=CHUCK ,ACCESS=U ,RESTRICT=VALUE ,NAME=IDSEG,
     VALUE=D.DIVISION EQ 'EAST' ,$
```

The prefix “D.” in front of the field DIVISION signals the use of the data file value of DIVISION. In this case, user Chuck can only change records of people who are in the ‘EAST’ division. If, instead, you use

```
VALUE=DIVISION EQ 'EAST'
```

for the value test, Chuck will be able to change any record he wants, but the only value he can enter for the DIVISION field is EAST.

The segment name on which the test is to be applied is given as the NAME parameter. If a request statement does not perform any action on this segment, the test itself is not performed. This is true even if you are making changes to a segment that is a child of the segment on which the test is performed.

The VALUE tests are added to any VALIDATE conditions that the MODIFY request contains. Only transactions passing both the VALIDATE and VALUE tests are accepted for processing.

### Restricting Both Read and Write Values

In many cases it will prove useful to issue both ACCESS=W (for MODIFY) and ACCESS=R (for TABLE) value restrictions for a user. This will both limit the values a user can write to the file and limit the data values that the user can actually see. You do this by issuing a RESTRICT=VALUE statement with an ACCESS=R to prohibit the user from seeing any values other than those specified in the test condition. You then issue a RESTRICT=VALUE statement with an ACCESS=W that specifies the write restrictions placed on the user. You cannot use ACCESS=RW to do this.

For example:

```
USER=TILLY ,ACCESS=R ,RESTRICT=VALUE ,NAME=IDSEG,
     VALUE=DIVISION EQ 'NORTH',$
            ACCESS=W ,RESTRICT=VALUE ,NAME=DIVTEST,
     VALUE=DIVISION EQ 'NORTH',$
```

**Note:** HLI requires ACCESS=RW.

## Placing Security Information in a Central File: The DBAFILE Attribute

The DBAFILE attribute enables you to place all of the passwords and restrictions for many Master Files in one central file. Each individual Master File points to this central control file. Groups of Master Files with the same DBA password may share a common DBAFILE which itself has the same DBA password.

There are several benefits to this technique. The primary ones are:

- Passwords only have to be stored once when they are applicable to a group of data files. This simplifies password administration.
- Files with different DBA passwords can now be JOINed or COMBINEd. In addition, individual DBA information remains in effect for each file in a JOIN or COMBINE.

The central DBAFILE is a standard Master File. Other Master Files can use the password and security restrictions listed in the central file by specifying its file name with the DBAFILE attribute. The syntax is

```
END
DBA=dbaname, DBAFILE=filename ,$
```

where:

*dbaname*
: Is the same as the dbaname in the central file.

*filename*
: Is the name of the central file.

You can specify passwords and restrictions in a DBAFILE that apply to every Master File that points to that DBAFILE; you can also include passwords and restrictions for specific Master Files by including FILENAME attributes in the DBAFILE. The following example shows a group of Master Files that share a common DBAFILE named FOUR:

```
ONE MASTER
FILENAME=ONE
  .
  .
END
DBA=ABC, DBAFILE=FOUR,$


TWO MASTER
FILENAME=TWO
  .
  .
END
DBA=ABC, DBAFILE=FOUR,$


THREE MASTER
FILENAME=THREE
  .
  .
END
DBA=ABC,
DBAFILE=FOUR,$


FOUR MASTER
FILENAME=FOUR,$
SEGNAME=mmmmm,$
FIELDNAME=fffff,$
END
DBA=ABC,$
   PASS=BILL,ACCESS=R,$
   PASS=JOE,ACCESS=R,$
FILENAME=TWO,$
   PASS=HARRY,ACCESS=RW,$
FILENAME=THREE,$
   PASS=JOE,ACCESS=R,RESTRICT=...,$
   PASS=TOM,ACCESS=R,$
```

**Note:** All Master Files that specify the same DBAFILE have the same DBA password.

The central DBAFILE is a standard Master File. It may include additional attributes before the END statement that signifies the presence of DBA information. The DBA password in the DBAFILE is the same as the password in all the Master Files that refer to it. This prevents individuals from substituting their own security. All of these Master Files should be encrypted.

**Note:**

- The DBAFILE may contain a list of passwords and restrictions following the DBA password. These passwords apply to all files that reference this DBAFILE. In the example above, PASS=BILL, with ACCESS=R (read only), applies to all files that contain the attribute DBAFILE=FOUR.
- After the common passwords, the DBAFILE may specify file-specific passwords and additions to general passwords. You implement this feature by including FILENAME attributes in the DBA section of the DBAFILE (for example, FILENAME=TWO). Consult *File Naming Requirements* on page 11-25 for additional information about the FILENAME attribute.
- File-specific restrictions override general restrictions for the specified file. In the case of a conflict, passwords in the FILENAME section take precedence. For example, a DBAFILE might contain ACCESS=RW in the common section, but specify ACCESS=R for the same password by including a FILENAME section for a particular file.
- Value restrictions accumulate; all value restrictions must be satisfied before retrieval. In the preceding example, note the two occurrences of PASS=JOE. JOE is a common password for all files, but in FILENAME=THREE it carries an extra restriction, RESTRICT=…, which applies only to file THREE.

## File Naming Requirements

When a DBAFILE includes a FILENAME attribute for a specific Master File, the FILENAME attribute in the referencing Master File must be the same as the FILENAME attribute in the DBA section of the DBAFILE. This prevents users from renaming a Master File to a name not known by the DBAFILE. For example:

```
ONE MASTER
FILENAME=XONE
  .
  .
  .
END
DBA=ABC, DBAFILE=FOUR,$


FOUR MASTER
FILENAME=FOUR
  .
  .
  .
END
DBA=ABC,$
  .
  .
  .
FILENAME=XONE,$
  .
  .
  .
```

ONE MASTER is referred to in requests as TABLE FILE ONE. However, both ONE MASTER and the DBA section of the DBAFILE, FOUR MASTER, specify FILENAME=XONE.

## Connection to Existing DBA System

If there is no mention of the new attribute, DBAFILE, there will be no change in the characteristics of an existing system. In the current system, when a series of files is JOINed, the first file in the list is the controlling file. Its passwords are the only ones examined. For a COMBINE, only the last file's passwords take effect. All files must have the same DBA password.

In the new system, the DBA sections of all files in a JOIN or COMBINE are examined. If DBAFILE is included in a Master File, then its passwords and restrictions are read. To make the DBA section of a file active in a JOIN list or COMBINE, specify DBAFILE for that file.

Once you start to use the new system, you should convert all of your Master Files. For database administrators who want to convert existing systems but do not want a separate physical DBAFILE, the DBAFILE attribute can specify the file itself. For example:

```
FILENAME=SEVEN,
  SEGNAME=..
    FIELDNAME=...
      .
      .
      .
END
DBA=ABC,DBAFILE=SEVEN,$    (OR DBAFILE= ,$)
 PASS=...
 PASS=...
```

## Combining Applications

Since each file now contributes its own restrictions, you can now JOIN and COMBINE files that come from different applications and have different DBA passwords. The only requirement is a valid password for each file. You can therefore grant access rights for one application to an application under the control of a different DBA by assigning a password in your system.

## Using Filters

You can assign screening conditions to a file that are automatically applied to any report request that accesses the file. See the *Creating Reports* manual for details.

## Summary of Security Attributes

The following is a list of all the security attributes used in FOCUS:

| Attribute | Alias | Maximum Length | Meaning |
|---|---|---|---|
| DBA | DBA | 8 | Value assigned is code name of the Database Administrator (DBA) who has unrestricted access to the file. |
| USER | PASS | 8 | Values are arbitrary code names, identifying users for whom security restrictions will be in force. |
| ACCESS | ACCESS | 8 | Levels of access for this user. Values are:<br>R read only<br>W write new segments only<br>RW read and write<br>U update values only |
| RESTRICT | RESTRICT | 8 | Types of restrictions to be imposed for this access level. Values are:<br>SEGMENT<br>FIELD<br>VALUE<br>SAME<br>PROGRAM<br>NOPRINT |
| NAME | NAME | 66 | Name of segment or field restricted or of the program to be called. |
| VALUE | VALUE | 80 | Test expression which must be true when RESTRICT=VALUE is the type of limit. |
| DBAFILE | DBAFILE | 8 | Names the Master File that contains passwords and restrictions to use. |

## Hiding the Restriction Rules: The ENCRYPT Command

Since the restriction information for a FOCUS database is stored in its Master File, you will want to encrypt the Master File in order to prevent users from examining the restriction rules. Only the Database Administrator can encrypt a description. Thus, you must set PASS=DBAname before you issue the ENCRYPT command. The syntax of the ENCRYPT command varies from operating system to operating system. See the *Overview and Operating Environments* manual for information on your operating system.

The following is an example of the complete procedure:

```
SET PASS=JONES76
ENCRYPT FILE PERS
```

The process can be reversed if you wish to change the restrictions. The command to restore the description to a readable form is DECRYPT.

The DBA password must be issued with the SET command before the file can be decrypted. For example:

```
SET PASS=JONES76
DECRYPT FILE PERS
```

## Encrypting Data

You may also use the ENCRYPT command within the Master File to encrypt some or all of its segments. When encrypted files are stored on their external media (disk or tape) they are secure from unauthorized examination.

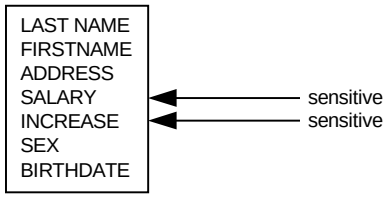
Encryption takes place on the segment level; that is, the entire segment is encrypted. The request for encryption is made in the Master File by setting the attribute ENCRYPT to ON. For instance:

```
SEGMENT=COMPSEG, PARENT=IDSEG, SEGTYPE=S1, ENCRYPT=ON,$
```

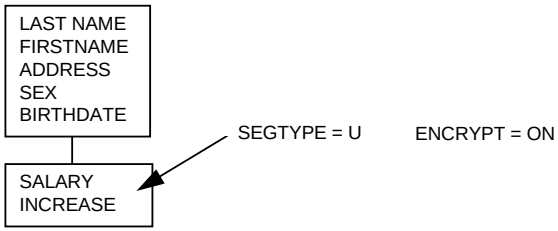
You must specify the ENCRYPT attribute before you enter any data in the file. The message “NEW FILE…” must appear when the encryption is first requested. Encryption cannot be requested later by a change to the Master File and cannot be removed once it has been requested and any data has been entered in the file.

### Performance Considerations

There is a small loss in processing efficiency when data is encrypted. You can minimize this loss by grouping the sensitive data fields together on a segment and making them a separate segment of SEGTYPE=U, unique segment, beneath their original segment. For example, suppose the data items on a segment are:




They should be grouped as:




## Restricting Existing Files

When you write a new Master File for a new file and include security limitations, data added to the data file is automatically protected according to those rules. If you write a new Master File for an already existing file that contains no data (that is, one in which the FOCUS MODIFY command produces the message “NEW FILE” or one for which a CREATE FILE command must be issued in MVS) the data will also be automatically protected. If, however, you have existing files to which you want to add security limitations, you need to use the RESTRICT command. (**Note:** This is not the RESTRICT attribute described in *Limiting the Access: The RESTRICT Attribute* on page 11-15.)

**Note:** The RESTRICT command cannot be used for non-FOCUS data sources.

# Displaying the Decision Table

When you enter security attributes for a Master File, FOCUS creates an internal decision table that lists users and their access privileges. You can display the decision table associated with a given file whenever the Master File of the file is not encrypted. Since you have to decrypt your Master File when you change or augment passwords, you can request a decision table picture to check your work. You can also decrypt your Master File to check your decision table. In FOCUS, after you have decrypted your file, type EX DBATABLE and provide the file name when prompted for it. For example:

```
>
ex dbatable
FILE NAME:
pers
>
>


 PAGE     1

 DATA BASE ADMINISTRATOR JONES76
 DBAFILE PERS      ON 01/18/94 AT 13.09.07

 FILENAME   USER      ACCESS  RESTRICT  NAME           VALUE
 --------   ----      ------  --------  ----           -----

 PERS
            BILL      R       SEGMENT   COMPSEG
            JOHN      R       FIELD     SALARY
                      R       FIELD     INCREASE
            LARRY     U       FIELD     SALARY
            MARY      W       VALUE     SALTEST        INCREASE+SALARY GE SALARY
                      W       VALUE     HISTTEST       DIV NE ' ' AND DATE GT 0
            TOM       RW
            TONY      R       VALUE     IDSEG          DIVISION EQ 'WEST'
```

The complete DBATABLE syntax is

```
EX DBATABLE filename, {LONG|SHORT}
```

where:

*filename*
: Is the name of the Master File for which you want a decision table.

LONG
: Displays 66 characters in the NAME column.

SHORT
: Display 18 characters in the NAME column.

**Note:** The DBATABLE procedure is supplied with FOCUS. Contact your system administrator if you cannot locate it.

## Setting Passwords Externally

Passwords can also be set automatically by an external security system such as RACF®, CA-ACF2®, or CA-Top Secret®. Passwords issued this way are set when FOCUS is first entered and may be permanent (that is, not alterable by subsequent SET USER, SET PASS or -PASS commands); or they may be default passwords that can be subsequently overridden; or they may be permanent for some users, defaults for other users, and not set at all for yet other users.

The advantage of setting FOCUS passwords externally is that the password need not be known by the user, does not require prompting, and does not have to be embedded in a PROFILE FOCEXEC or an encrypted FOCEXEC.

Passwords set this way must match the passwords specified in the Master Files of the files being accessed.

See the *FOCUS for IBM Mainframe Installation Guide for CMS* or *TSO* and the *FOCUS for IBM Mainframe Multi-Session Option Installation and Technical Reference Guide*, for FOCUSID installation instructions.

# FOCEXEC Security

Most data security issues are best handled by the FOCUS DBA facility. Nevertheless, some additional data security facilities are incorporated within Dialogue Manager. These are:

- Suppressing password display.
- Setting passwords in encrypted FOCEXECs.
- Defining variable passwords.
- Encrypting and decrypting FOCEXECs.
- Locking FOCEXEC users out of FOCUS.

External security systems can also set passwords through the FOCUSID exit routine.

## Suppressing Password Display

The NODISPLAY attribute can be used within -CRTFORM to create a password prompt with no display of the input characters. The syntax is:

```
<.NODISP.&mypass
```

Consider the following example, in which the attribute .NODISP before the variable instructs the system to accept the response, but not display it, and to set the password to the value that was altered:

```
-SET &MYPASS = '12345678' ;
-CRTFORM
-" ENTER YOUR PASSWORD <.NODISP.&MYPASS "
SET PASS = &MYPASS
```

## Setting Passwords in Encrypted FOCEXECs

Passwords can be set within FOCEXECs and tied to different portions of FOCEXECs according to this syntax:

```
-PASS password
```

Since -PASS is a Dialogue Manager statement, it executes immediately and is not sent to the FOCSTACK. This means that the user need not issue the password with the SET command. It also means that the password is not visible to anyone. Of course, the procedure must be encrypted so that the password cannot be revealed by printing the procedure.

## Defining Variable Passwords

The Dialogue Manager command -PASS can have a variable attached to it as well as a literal. The syntax is:

```
-PASS &value
```

For example:

```
-PASS &MYPASS
-PASS &VAL.ENTER YOUR PASSWORD.
```

This command is only visible when you edit the FOCEXEC. It does not appear when the ECHO option is ALL and is not printed in a batch run log.

## Encrypting and Decrypting FOCEXECs

You may want to keep the actual text of a stored FOCEXEC confidential while allowing users to execute the FOCEXEC. You may want to do this either because there is confidential information stored in the FOCEXEC or because you do not want the FOCEXEC changed by unauthorized users. You can protect a stored FOCEXEC from unauthorized users with the ENCRYPT command.

Any user can execute an encrypted FOCEXEC, but you must decrypt the FOCEXEC to view it. Only a user with the DBA password can decrypt the FOCEXEC.

You use the following procedure to encrypt the FOCEXEC named SALERPT:

```
SET PASS = DOHIDE
ENCRYPT FILE SALERPT FOCEXEC
```

Anyone can execute the FOCEXEC by typing EX SALERPT. The FOCEXEC can only be viewed by decrypting it, as follows:

```
SET PASS = DOHIDE
DECRYPT FILE SALERPT FOCEXEC
```

Encrypted FOCEXECs cannot be echoed (that is, have their commands displayed on the terminal), so &ECHO has no effect.

## Locking FOCEXEC Users Out of FOCUS

Users can normally respond to a Dialogue Manager value request with QUIT and return to the FOCUS command level. In situations where it is important to prevent users from entering or returning to FOCUS, the environment can be locked and QUIT can be deactivated by entering in a FOCEXEC:

```
-SET &QUIT=OFF;
```

With QUIT deactivated, any attempt to leave the Dialogue Manager environment will produce an error message. Following the error message, the user will be reprompted for the needed value.

A user may still terminate the session from inside a locked environment by responding to a prompt with:

```
QUIT FOCUS
```

This returns the user to the operating system, not to the FOCUS command level.

The default setting for &QUIT is ON.

# Program Accounting/Resource Limitation

In addition to controlling access to a database, FOCUS security features can be used for program accounting and limiting the amount of computer resources a given user can use. When you specify RESTRICT=PROGRAM in the Master File, you automatically call a user-written program to monitor various people's use of the database. You can also use value tests to limit the number of records that can be requested, thus limiting waste that may result from a user requesting unwanted data.

Additionally, the Usage Accounting and Security Exit Routine (UACCT) provides information on usage statistics and attempted violations to FOCUS database security, as well as to external security systems.

## Program Accounting

You can use FOCUS security attributes to specify that a user-written program be called immediately after a TABLE or GRAPH command has completed, but before the report is printed. You activate this user-written program by assigning the value PROGRAM to the RESTRICT attribute. The program is passed the statistics of the run (that is, number of records retrieved, lines of sorted results) and the identity of each data field that was active in the run. You can use this user program exit to:

- Monitor the retrieval activity for particular files. For instance: number of requests, number of records retrieved, distribution of usage by user category (by userid or password identity).
- Monitor the usage frequency of items in a database.
- Perform usage accounting based on values such as number of records retrieved.
- Provide another level of security in which the user program exit determines whether the report should be displayed.

The accounting aspects of this feature are enforceable only for the TABLE and GRAPH commands, not the TABLEF command.

## Activating a DBA User Program

You activate a DBA user program by adding the following attributes to the security section of any Master File for each user that you want the program to monitor

```
USER=name, ACCESS=R, RESTRICT=PROGRAM,
NAME=pgmname, VALUE=returncode,$
```

where:

*name*
: Is the arbitrary code name used to identify the user (8 bytes).

*pgmname*
: Is the name of the user-written program (8 bytes).

*returncode*
: Must be matched with the DBA name (8 bytes).

For example:

```
USER=PETER, ACCESS=R, RESTRICT=PROGRAM,
NAME=PETER1, VALUE=D76,$
```

You can specify other restrictions for the users mentioned in addition to calling the program.

## Specifications for the User-Written Program

The user program must be coded as a subroutine in a language which can be dynamically linked at FOCUS execution time in the operating environment. COBOL is acceptable in all environments, as are PL/I and Assembler. Languages acceptable for different environments are covered in the *Overview and Operating Environments* manual.

Six arguments are supplied to the user program. The first five of these are computed by FOCUS, the last is returned by the user program to FOCUS. The value of the last argument is matched to a value provided in the DBA section of the Master File. The purpose of this is to prevent a spurious program of the same name from being substituted for the real one. If the DBA value and the retrieval value do not match, the report is not printed and FOCUS exits immediately.

The arguments to the call are:

| Argument | Format | Length | Description |
|---|---|---|---|
| FILEID | Alpha | 18 bytes | The name of the file. |
| NUMB | Int | 4 bytes | The number of data and defined fields in the file. |
| ACT | Bit String | 8 byte units | Each bit is associated with a data field. A value of 1 means active for the request. |
| RECORDS | Int | 4 bytes | Number of records retrieved. |
| LINES | Int | 4 bytes | Number of records (not including options such as headings, footings, and page numbers) to be printed. |
| RETVALUE | Alpha | 8 bytes | Returned by user program to be matched with DBA-supplied value. |

## Resource Limitation

You can make a VALUE condition for some overall limitation on retrieval ability. For instance, you can limit the maximum number of records a user can retrieve in a single TABLE request. This restriction can be activated if a selected segment is referred to in the request, or it can be active for every request.

Record limitation is added by the phrase

```
RESTRICT=VALUE ,NAME= {segname|SYSTEM} ,VALUE=RECORDLIMIT EQ n,$
```

where:

*n*
: Is an integer greater than 0.

For example:

```
USER=TILLY, ACCESS=R, RESTRICT=VALUE, NAME=SYSTEM,
VALUE=RECORDLIMIT EQ 1000, $
```

or

```
USER=TILLY, ACCESS=R, RESTRICT=VALUE, NAME=COMPSEG,
VALUE=RECORDLIMIT EQ 1000, $
```

The second example will limit the number of records retrieved only if fields from segment COMPSEG are referred to in the report request.

For external files, READLIMIT EQ can be used exactly as RECORDLIMIT EQ to set an automatic maximum on the number of successful reads issued for sequential files or the number of calls made to an external file system.

## Usage Accounting and Security Exit Routine (UACCT)

The Usage Accounting and Security Exit Routine (UACCT) provides information for an installation:

- To log FOCUS usage after FOCUS commands which access data, such as TABLE, MODIFY, or MATCH.
- To capture attempted violations of the DBA provisions in the Master File.
- To trap violations detected by external security systems.

The distributed copy of FOCUS contains a dummy version of the UACCT exit routine. To use a working version of UACCT, you must install it as described in the Installation Guides for CMS and TSO and the *FOCUS for IBM Mainframe Multi-Session Option Installation and Technical Reference Guide*.

# Absolute File Integrity

FOCUS can perform shadow paging to guarantee the integrity of any FOCUS file created. This option does require extra disk space, so it is up to the Database Administrator to decide whether Absolute File Integrity is necessary for the file.

FOCUS shadow paging is accomplished by checkpoints and directory pages, which, in one stroke, change the shadow pages into current database pages. Basically, FOCUS creates a shadow image of a FOCUS file, with each FOCUS page having a corresponding shadow page. At any point, one of the database images has complete data integrity, regardless of what happens to the other. Therefore, the data integrity of a FOCUS file will never be compromised by a system crash or other circumstances.

Absolute File Integrity is available for FOCUS files in all operating system environments. (Because CMS automatically provides shadow paging, invoking the FOCUS facility for Absolute File Integrity is generally not necessary under CMS.)

**Note:** IBM no longer guarantees data integrity for file mode A6 as of VM/SP6. FOCUS can still shadow the file correctly but IBM does not guarantee integrity on these files.

To invoke Absolute File Integrity in FOCUS, before creating the file with the CREATE FILE command issue the following command

```
SET SHADOW = value
```

where:

*value*

Can be one of the following:

OFF does not invoke Absolute File Integrity. This is the default value.

ON invokes Absolute File Integrity.

OLD invokes the use of the shadow technology available in FOCUS releases prior to 7.0 (before the size limit for FOCUS files was increased). This means fewer pages are shadowed. If your FOCUS file was created with this option, the maximum number of pages is 63,551. If this limit is exceeded, FOCUS displays the (FOC198) error message.

If the file has already been created, take the following steps to invoke Absolute File Integrity:

1. Specify the REBUILD, REORG, and DUMP options with the REBUILD command.
2. Invoke the Absolute File Integrity facility with the SET SHADOW=ON command.
3. Create the FOCUS file with the CREATE FILE command.
4. Specify the REBUILD, REORG, and LOAD options with the REBUILD command.

# 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 OF FOCUS    FILE EMPLOYEE ON 01/05/96 AT 14.02.35
        I SECSEG          I SKILLSEG
 05     I EMPINFO   06    I KL
 01     I KLU
        I S1
        .............     .............
        *************     :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 can be 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 User Exits for Non-FOCUS Data Sources

**Topics:**

- The Dynamic and Re-Entrant Private User Exit of the FOCSAM Interface
- User-coded Data Access Modules
- Re-Entrant VSAM Compression Exit: ZCOMP1

This appendix describes three ways to read non-FOCUS data sources with user-written procedures.

## The Dynamic and Re-Entrant Private User Exit of the FOCSAM Interface

The FOCSAM Interface contains a user exit that can be invoked as an alternative to the lowest-level retrieval routines that are part of the Interface. This interface is responsible for data access to VSAM and QSAM structures. The Master File would specify SUFFIX=VSAM or SUFFIX=FIX. This exit makes it possible to combine user-written code, devoid of any dependence on internal FOCUS structures, with the logical retrieval functions of the FOCSAM Interface, such as record selection logic, treatment of missing records in multi-record files, JOINS between various types of files, and so forth. This exit is an alternative to the original GETPRV Private exit. It is not upwardly compatible with the original exit documented in Tech Memo 7767, Tech Memo 7767.1, or Tech Memo 7767.2. It would require coding changes for reentrancy, additional functions, and Access File requirements. The original GETPRV exit continues to be supported as documented in the aforementioned Tech Memos. The major differences are as follows:

1. By providing a new parameter, CONTEXT, the exit supports reentrancy, providing the benefit of reduced storage requirements as well as enhanced invocation performance.
2. Support for multiple concurrent exit processors is provided through an Access File where a user exit module can be named on a per Master File basis.
3. The user exit is dynamically called at execution time, unlike the original GETPRV exit that required a link to either VVSET or FOCSAM, thus avoiding the need to modify FOCUS after each upgrade or application of maintenance. There is no more need for link-edits to FOCUS.

4. An initialization call has been added to allow the exit code to perform initial housekeeping.
5. The QUALIF parameter supports the additional options of (O) OPEN file, (R) OPEN request (position), (C) CLOSE, and (F) Fin of FOCUS. These new control options are in addition to the (S), (G), and (E) read options that have always been available in the original GETPRV exit and eliminate the separate PRVCLS call of the original exit as well.
6. The new exit supports multiple positions on the same file.

## Functional Requirements

Functionally, the private code is a substitute for retrieval calls typically used against, but not limited to, key-sequenced VSAM files and can be used against any data source that can be represented as such a file. The private code need not deal with any intra-record structures represented by OCCURS clauses, nor with the translation of FOCUS IF conditions into lower-level criteria. Both of these functions are performed by the driving logic in the FOCSAM Interface.

The user-written code must be able to do the following:

1. Obtain a record, given a full value of the key. The key is presumed to be unique (direct read).
2. Obtain a record that is greater than or equal to a given value of the full or partial key (generic read).
3. Obtain the next logical record, starting from the current position in the file (sequential read). Successive sequential reads must return records that are in ascending sequence (bit by bit) on the key field.
4. Direct and generic reads that retrieve records must establish the starting position in the file for subsequent sequential reads. Direct reads that fail to retrieve the requested records need not establish these positions.
5. For the open file request (O), it must logically or physically open the file.
6. If the logical end-of-file is reached as a result of a sequential read, an end-of-file signal must be returned. Subsequent sequential reads must return end-of-file signals rather than error indications, for example, when processing a JOIN.
7. A ‘close’ function must be provided that results in the release of all resources and loss of position in the file, so that subsequent open requests succeed.
8. Successive close calls must be innocuous. Be prepared to close a file that is already closed.
9. A unique area must be obtained, for example, GETMAIN, and maintained using the CONTEXT parameter on a per DDNAME basis.
10. The code must be serially reusable and reentrant. It must be linked AMODE 31.

### Installation Note

This exit is only supported with FOCSAM=NEW. There is no need to link the user written exit with VVSET or FOCSAM as was required with the original GETPRV exit.

## Implementation

The Dynamic GETPRV Exit is linked as a separate module and loaded or called from FOCUS.

### The Master File

The Master File for data to be accessed using this user exit is exactly the same as the description of any other data source read by the FOCSAM Interface except that the SUFFIX must specify PRIVATE. All other READ ONLY features of the FOCSAM Interface are fully supported.

*Example* **Sample Master File**

```
FILE=filename,  SUFFIX=PRIVATE,$
SEGNAME=ROOT ,  SEGTYPE=S0,$
  GROUP=keyname   , ALIAS=KEY       , USAGE=xx, ACTUAL=xx  ,$
  FIELD=fieldname1, ALIAS=aliasname1, USAGE=xx, ACTUAL=xx  ,$
  FIELD=fieldname2, ALIAS=aliasname2, USAGE=xx, ACTUAL=xx  ,$
```

**Note:** SUFFIX=PRIVATE. No other options will invoke the exit.

### The Access File

An Access File is required and provides a pointer to the actual name of the private exit. The PDS used for the Access File must be allocated to DDNAME ACCESS.

*Example* **Sample Access File**

```
MODNAME=pgmname,$
```

*Example* **Allocating an Access File**

```
//ACCESS  DD DSN=access.file.pds.name,DISP=SHR
```

**Warning:** If no Access File exists or DDNAME ACCESS is not allocated, FOCUS assumes that you are using the old GETPRV exit and that all of the original GETPRV rules are in affect. In this case, this technique does not apply.

## Calling Sequence

The user-coded retrieval routine is written as a standalone program. There are no limitations to program name other than standard IBM rules. We recommend that the program be written in a language that fully supports reentrancy, such as ASSEMBLER or C. The parameter list is as follows:

`NCHAR`

Full-word binary integer, posted by the exit. A positive number indicates the length in bytes of the record obtained; zero indicates no record or end-of-file (there is no difference). Must be non-zero for every successful read. Used by read options (S), (E), and (G). Do not modify the pointer. Instead, modify the location addressed by the pointer.

`DDNAME`

8-byte character argument, posted by FOCSAM, corresponding to the FOCUS filename for the generated report. For example, TABLE FILE CAR results in DDNAME CAR, left-justified and blank-padded. Used for all options except (F). Do not modify this parameter.

`ABUFF`

Full-word binary integer, posted by the exit; the absolute address of the record obtained by this call. Used with all read options (S), (E), and (G). Do not modify this pointer. Instead, modify the location addressed by the pointer.

`RC`

Full-word binary integer return code, posted by the exit. Zero indicates no error: non-zero indicates some type of error. Used with all options. Do not modify the pointer. Instead, modify the location addressed by the pointer.

`KEY`

Full-word binary integer posted by FOCSAM, containing the full value of the key for direct or generic reads. Not significant for sequential reads. The key value is left justified and less than 255 bytes long.

Note that the exit must know the true key length and its format. Used with options (E) and (G). Do not modify this parameter.

`OPT`

Four-byte character argument, posted by FOCSAM, indicating the type of call. Do not modify this parameter. The OPTions are as follows:

READ OPTIONS

'S ' =sequential read.

'E ' =direct read (EQ).

'G ' =generic read (GE).

CONTROL OPTIONS

'O ' =OPEN file.

'R ' =OPEN request (position) used for recursive JOINs.

'C ' =CLOSE file.

'F ' =FIN for FOCUS -- final housekeeping should be done here.

These control and read arguments must include three trailing blanks.

`CONTEXT`

Full-word binary integer, posted by FOCSAM, which points to a work area described below. Do not modify this parameter.

**Note:** The parameters passed to the exit are not delimited with the final parameter having the high-order bit set on. Make sure that your program does not scan for this high-order bit.

## Work Area Control Block

`EYECATCH`

An 8 character string containing the literal 'PRIVATE '.

`PFMCB`

Full-word binary integer, posted by the exit. Generally set during OPTion (O) processing by the exit program and returned unchanged by subsequent FOCSAM calls. This parameter is generally used to point to the dynamic work areas used to maintain reentrancy.

`PFACB`

Full-word binary integer, posted by the exit. Generally set during OPTion (O) processing by the exit program and returned unchanged by subsequent FOCSAM calls for OPTions (C), (R), (E), (G), and (S) and is unique by DDNAME. This is generally used as a physical file context. This parameter is not valid for OPTion (F).

`PFRPL`

Full-word binary integer, posted by the exit. Generally set during OPTion (R) processing by the exit program and returned unchanged by subsequent FOCSAM calls for OPTions (E), (G), and (S) and is unique for each view within the above PFACB parameter. This is generally used for logical file context.

This parameter is not valid for OPTion (F).

`KEYLENF`

Full-word binary integer, posted by the exit. Generally set during OPTion (O) processing by the exit program and should contain the whole key length for the file.

`KEYLENR`

Full-word binary integer, posted by the exit. Generally set during OPTions (G) and (E) processing by the exit program and should contain the actual key length for a direct read.

`ERRTEXT`

Full-word binary integer, posted by the exit. Should contain the absolute starting address of a message. FOCUS displays this message if the RC parameter returned by the exit is non-zero and for OPTions (E), (G), and (S).

`ERRTEXTL`

Full-word binary integer, posted by the exit. It should contain the length of the above ERRTEXT message.

`INDEX`

Full-word binary integer, posted by FOCSAM.

This option contains the index # by which to access the file.

| | |
|---|---|
| 0 | Primary key in Master File in Master -- KEY-DKEY |
| 1,2,e | Secondary indexes in the Master File |
| tc | KEY1-DKEY1 or KEY2-DKEY2, and so on, and INDEX=I |

`RESERVE1`

Full-word binary integer, reserved for future use.

`USERID`

Full-word binary integer, posted by FOCSAM. This userid will only be present if found in the central site security environment.

`RESERVE2`

Full-word binary integer, reserved for future use.

***Example*** **Sample Assembler DSECT**

```
GETPRVPL DS    0F
NCHAR@   DS    A
DDN@     DS    A
ABUF@    DS    A
RC@      DS    A
KEY@     DS    A
OPT@     DS    A
CONTEXT@ DS    A
         TITLE 'GETPRV CONTEXT'
CONTEXTD DSECT
EYE      DS    CL8   eye catcher literal
PFMCB@   DS    A     handle
PFACB@   DS    A     file open handle
PFRPL@   DS    A     file retrieval handle
KEYLEN_F DS    F     key length for file
KEYLEN_R DS    F     key length for this request
RETTEXT@ DS    A     pointer to returned message
LRETTEXT DS    F     length of returned message
INDEX    DS    F     index for file access - 0 primary  1... secondary
         DS    A     reserved
USERID   DS    CL8
         DS    2F    reserved
         SPACE 1
```

***Example*** **Sample C Declaration Required For Invoking FFUN Function**

```
/*
control block for additional info
*/
typedef struct getprv_inf_s {
char         eye[8];     /* I: eye catcher "PRIVATE "*/
Pvoid        pfmcb;      /* o: p' to handle forgetprv   */
                         /*    set up by user at first option O */
                         /* I: p' to handle for getprv   */
                         /*    passed to user by all other calls*/
Pvoid        pfacb;      /* o: p' to handle for file */
                         /*    set up by user at option O       */
                         /* i: p' to handle for file */
                         /*    passed to user at option C,R,E,G,S */
Pvoid        pfrpl;      /* o: p' to handle for request  */
                         /*    set up by user at option R       */
                         /* i: p' to handle for request  */
                         /*    passed to user at option E,G,S */
long         keylen_fil; /* I: key length (whole) for the file . */
                         /*    used at option O . */
long         keylen_req; /* I: key length for the direct read request*/
                         /*    used at direct read options  G,E  */
char        *rettext;    /* O: a'native db error msg text */
long         lrettext;   /* O: l'native db error msg text */

long         index;      /* I:index # by which to access file :
                             = 0    - primary key
                                      in master - KEY|DKEY
                             = 1,2,... - secondary indexes
                                      in master - KEY1|DKEY1 or KEY2|DKEY2 ...
                                      and INDEX=I */
long         res1[1];    /* reserved */
char         userid[8];  /* user id */
long         res2[2];    /* reserved* /
} getprv_inf_t;
/*

*/
typedef  void FFUN getprv_t (
 long       *nchar      /* out: length of data record read. =0  */
                        /*      if eof or no rec found         {*/
                        /*     used in all read options S,E,G   */
,char       *ddn        /* in : ddname to read    */
                        /*     used in all options except F     */
,char       **abuf      /* out: a' buffer */
                        /*     used in all read options S,E,G   */
,long       *rc         /* out" return code. = 0 if ok */
                        /*     used in all options         */
,char       *key        /* in: key value for read */
                        /*     used in read options E,G   */
```

```
,char       *opt        /* in: read option : */
                        /*    S sequential read */
                        /*    G     GE read */
                        /*    E     EQ read */
                        /*     control options */
                        /*    O  open file */
                        /*    R  open request (position)*/
                        /*    C  close file */
                        /*    F  fin of focus */
,getprv_inf_t  *        /* in/out other info .see above */
);
#endif
```

# User-coded Data Access Modules

A user routine may be written to provide records to the FOCUS report writer from any non-standard data source that cannot be directly described to FOCUS using the Master File. The record, which can come from any data source, is treated by FOCUS exactly as if it had come from a FOCUS database. The user routine must be coded as a subroutine in FORTRAN, COBOL, BAL, or PL/I, and must pass data to the FOCUS calling program through arguments in the subroutine.

The user program is loaded automatically by the report writer and is identified by the file suffix, in the Master File. For example, if the Master File contains:

```
FILE = ABC, SUFFIX = MYREAD
```

FOCUS will load and call the program named MYREAD to obtain data whenever there is a TABLE, TABLEF, MATCH, or GRAPH command against file ABC.

The record returned to FOCUS must correspond to a segment instance in the Master File. The layout of the fields in this record corresponds to the ACTUAL formats specified in the Master File for each segment.

It is the responsibility of the user program to determine which segment to pass to FOCUS if the Master File contains more than one segment. FOCUS will traverse the hierarchy in a top-to-bottom, left-to-right sequence; if the user routine can anticipate which segment FOCUS expects to be given, the number of rejected segments will decrease and the retrieval efficiency will increase.

In FORTRAN, the subroutine MYREAD would be coded as follows:

```
SUBROUTINE MYREAD (LCHAR, BUF, OFFSET, RECTYP, NERRX, CSEG, REGI, NFIND, MATCH,
IGNOR, NUMFLD, NUMLEV, CVT)
```

where:

`LCHAR`

(4 byte integer) If LCHAR > 0, LCHAR is the number of characters in the record passed back to FOCUS. If LCHAR = 0, the user routine is telling FOCUS that an end-of-file has been encountered and the retrieval is complete. If LCHAR = -N, the user routine is telling FOCUS that the buffer contains an error message of length N, to be printed out, and that the user routine should not be called again. LCHAR = -1 is reserved for Information Builders use.

`BUF`

This parameter is a 4096 byte buffer provided by FOCUS to receive the record from the user routine.

`OFFSET`

(4 byte integer) If the user routine puts data in BUF, OFFSET should be set to 0 each time the user routine is called. If the user routine provides its own buffer or buffers, OFFSET is the address of the user's buffer minus the address of BUF. A utility called IADDR is provided to compute an address. In FORTRAN, for example, one could code:

```
OFFSET = IADDR (MYBUF) - IADDR (BUF)
```

`RECTYP`

(4 byte integer) The number of the FOCUS segment corresponding to the record being presented to FOCUS, set by the routine. These numbers correspond to either the list obtained by issuing '? FILE filename' or the field SEGNO resulting from a CHECK FILE filename HOLD.

`NERRX`

(4 byte integer) Flag set by FOCUS. If NERRX < 0, FOCUS is directing the user routine to shut down (for example, close all files) and not provide any more records. On return, FOCUS will not call the subroutine again.

`CSEGX REGI NFIND MATCH`

Reserved for Information Builders use.

`IGNOR`

(4 byte integer) FOCUS will reject any segment whose number (see RECTYP) is greater than or equal to IGNOR. IGNOR is set by FOCUS, based on the segments referenced in the request, but may be reset by the user routine.

`NUMFLD NUMLEV`

Reserved for Information Builders. Use CVT.

On CMS, FOCUS will look for MYREAD TEXT on any accessed CMS minidisk. If MYREAD TEXT is not found, FOCUS will then look for a member MYREAD within all TXTLIB files currently pointed at by a GLOBAL TXTLIB command.

On TSO, the MYREAD object code, and any other routine that it calls, must be link-edited into a load module whose member name in the load library is MYREAD. This load library can be allocated as ddname USERLIB or concatenated with the STEPLIB program library.

# Re-Entrant VSAM Compression Exit: ZCOMP1

This re-entrant ZCOMP1 exit works with compressed VSAM and sequential files. It requires an initial call to ZCOMP0 for initial housekeeping and a USERWD parameter to anchor storage.

## Overview

The ZCOMP1 user exit was designed to provide an exit that can be user-supplied in order to decrypt coded fields, expand compressed records, and accomplish any other type of user-specific data manipulation required. This exit is designed to access all files readable by the FOCSAM Interface. It is the user's responsibility to write and maintain this exit.

There are no special Master File requirements. SUFFIX can equal VSAM (for KSDS or ESDS files), FIX (for sequential files), or PRIVATE (for file access through the GETPRV user exit).

## Linking ZCOMP1

After you write a ZCOMP1 user exit, it must be linked with VVSET. This process of linking ZCOMP1 into VVSET can be accomplished by using the GENFSAM EXEC (for VM), or the GENFSAM JCL (for MVS) that is found in the FOCCTL.DATA data set. Note that GENFSAM is designed to link in both ZCOMP1and GETPRV user exits at the same time. Should you only be implementing one of them, the VM EXEC generates the correct linkage to the routine required, whereas in the MVS JCL, the appropriate lines in the GENFSAM member must be commented out (these would be the INCLUDE OBJECT statements). ZCOMP1 can be linked re-entrant if you plan to use the USERWD parameter.

## What Happens When You Use ZCOMP1

The FOCSAM Interface reads the record for the allocated data source. Upon a successful read, FOCSAM calls ZCOMP0, if it exists, with the parameters listed below so that initial housekeeping can be performed. All subsequent calls are to ZCOMP1 with the same parameter list.

The ZCOMP1 exit is responsible for determining what to do with the parameter information it receives. The DDNAME can be used to determine whether the associated data source needs to be decompressed or not. If not, the user exit typically moves A(IRECLEN) to A(ORECLEN) and A(A(IREC)) to A(A(OREC)) and returns to FOCSAM with a zero A(STATCODE). If decompression or any other processing is required, it is the responsibility of the user exit to do so.

After the user exit has completed its processing, it should return with either the A(ORECLEN), A(A(OREC)) and a zero status code or with a non-zero status code which gives the following message:

```
(FOC1150) ZCOMP DECOMPRESS ERROR: status
```

**Note:** This error terminates a TABLE request.

## ZCOMP1 Parameter List

| Parameter | Description | Length |
|---|---|---|
| A(STATCODE)* | Pointer to fullword binary status code | 4 byte integer |
| A(DDNAME) | Pointer to 8 byte file name in use | 8 byte character |
| A(USERID) | Reserved for future use | 8 byte character |
| A(IRECLEN) | Pointer to length of original record | 4 byte integer |
| A(A(IREC)) | Pointer to pointer to original record | 4 byte integer |
| A(ORECLEN)* | Pointer to length of revised record | 4 byte integer |
| A(A(OREC))* | Pointer to pointer to revised record | 4 byte integer |
| A(USERWD)** | Pointer to fullword | 4 byte integer |

* The user supplies these parameters.
** This parameter can be used to anchor user storage for re-entrant processing.
**Note:** Never modify the primary pointers, but rather the pointers or values they point to.

Note that upon entry to ZCOMP1, the ORECLEN and OREC parameters are NULL. It is the responsibility of the user to fill these in correctly.

While processing the FOCUS FIN command, a call is placed to the ZCOMP2 entry point, which provides the user with the ability to do any other global cleanup required.

The parameters returned by ZCOMP1 are not validated. It is the responsibility of the user routine to ensure that valid addresses and lengths are returned to FOCUS from ZCOMP1.

Unpredictable results occur if incorrect parameters are passed back from the routine.

# D Rounding in FOCUS

**Topics:**

- Data Storage and Display
- Rounding in Calculations and Conversions

This appendix describes how FOCUS numeric fields store and display data, how rounding occurs in calculations, and what happens in conversion from one format to another.

## Data Storage and Display

Values are rounded before storage or before display, depending on the format. Integer fields (format I) and packed decimal fields (format P) are rounded before they are stored. Floating-point fields (formats F and D) are stored as entered and rounded for display.

When a final decimal digit is less than 5, the data value rounds down. A data value with a final digit of 5 or greater rounds up. The following rounding algorithm is used:

**1.** The incoming value is multiplied by 10.

**2.** This multiplication repeats the same number of times as the number of decimal places in the target format. For example, if 123.78 is input to a packed decimal field with one decimal place, it is multiplied by 10 once:

```
1237.8
```

**3.** Next, 0.5 is added if the incoming value is positive or subtracted if the incoming value is negative:

```
1237.8 + 0.5 = 1238.3
```

or, if the input value was -123.78,

```
-1237.8 - 0.5 = -1238.3
```

**4.** The value is truncated, and the decimal is shifted to the left.

```
123.8
```

or, if the original value was negative,

```
-123.8
```

The following chart illustrates the rounding differences between FOCUS numeric field formats. Subsequent topics discuss these differences in detail.

| Format | Type | Format | Input | Stored | Display |
|---|---|---|---|---|---|
| I | Integer | I3 | 123.78 | 0124 | 124 |
| F | Floating-Point<br>Single-Precision | F3<br>F5.1 | 123.78<br>123.78 | 123.7800<br>123.7800 | 124<br>123.8 |
| D | Floating-Point<br>Double-Precision | D3<br>D5.1 | 123.78<br>123.78 | 123.780000000000<br>123.780000000000 | 124<br>123.8 |
| P | Packed | P3<br>P5.1 | 123.78<br>123.78 | 0000124<br>00001238 | 124<br>123.8 |

## Integer Fields: Format I

An integer value entered with no decimal places is stored as entered.

When a value with decimal places is entered into an integer field using a transaction, that value is rounded, and the result is stored. If the fractional portion of the value is less than 0.5, the value is rounded down; if the fractional portion of the value is greater than or equal to 0.5, the value is rounded up. For example, if a value entered in a CRTFORM is 123.78, the value stored is 124.

However, if an integer field is computed, the decimal portion of the resulting value is truncated, and the integer portion of the answer is stored (or printed). For example, if the result of a calculation is 123.78, the value stored is 123.

# Floating-Point Fields: Formats F and D

Format type F describes single-precision floating-point numbers stored internally in 4 bytes. Format F is comparable to COBOL COMP-1. Format type D describes double-precision floating-point numbers stored internally in 8 bytes. Format D is comparable to COBOL COMP-2.

Formats F and D store as many decimal places as are input, up to the limit of the field's storage. Floating-point fields are stored in a logarithmic format. The first byte stores the exponent; the remaining 3 or 7 bytes store the mantissa, or value.

When the number of decimal places input is greater than the number of decimal places specified in the format, F and D field values are stored as they are input, up to the limit of precision. These values are rounded for display according to the field format. For example, if 123.78 is entered in a floating-point field with one decimal place, 123.78 is stored, and 123.8 is displayed.

Format D is more accurate than format F for larger numbers, since D fields can store up to 15 significant digits, and F fields are not accurate beyond 8 digits.

For floating-point fields (format D or F), the stored values of decimal numbers are in hexadecimal and may convert to a value very slightly less than the actual decimal number. When the final digit is 5, these numbers may round down instead of up.

# Packed Decimal Format: Format P

In packed-decimal format (format type P), each byte contains two digits, except the last byte, which contains a digit and the sign (D for negative numbers, C for positive). Packed fields are comparable to COBOL COMP-3.

Packed field values are rounded to the number of digits specified in the field format before they are stored.

When the number of decimal places input is greater than the number that can be stored, P field values are rounded first, then stored or displayed.

Packed fields are precisely accurate when sufficient decimal places are available to store values. Otherwise, since values are rounded before being stored, accuracy cannot be improved by increasing the number of digits displayed. For example, if 123.78 is input to a packed field with 1 decimal place, 123.8 is stored. If the field's format is then changed to P6.2 using a COMPUTE or DEFINE, 123.80 will be displayed. If the field's format is changed to P6.2 in the Master File, 12.38 is displayed.

***Example*** **Storage and Display**

For floating-point fields (format D or F), the stored values of decimal numbers are in hexadecimal and may convert to a value very slightly less than the actual decimal number. When the final digit is 5, these numbers may round down instead of up.

The following example shows an input value with two decimal places, which is stored as a packed field with two decimal places, a packed field with one decimal place, and a D field with one decimal place:

Master File:

```
FILE=FIVE, SUFFIX=FOC
 SEGNAME=ONLY, SEGTYPE=S1,$
  FIELD=PACK2,,P5.2,$
  FIELD=PACK1,,P5.1,$
  FIELD=DOUBLE1,,D5.1,$
```

Program to Load Data:

This MODIFY creates a file with three fields: a P field with two decimal places, a P field with one decimal place, and a D field with one decimal place. The same data values are then loaded into each field.

```
CREATE FILE FIVE

MODIFY FILE FIVE
FIXFORM  PACK2/5 PACK1/5 DOUBLE1/5
MATCH PACK2
  ON MATCH REJECT
  ON NOMATCH INCLUDE
DATA
1.05 1.05 1.05
1.15 1.15 1.15
1.25 1.25 1.25
1.35 1.35 1.35
1.45 1.45 1.45
1.55 1.55 1.55
1.65 1.65 1.65
1.75 1.75 1.75
1.85 1.85 1.85
1.95 1.95 1.95
END
```

TABLE Request:

This TABLE request prints the values and a total for all three fields.

```
TABLE FILE FIVE
PRINT PACK2 PACK1 DOUBLE1
ON TABLE SUMMARIZE
END
```

The following report results:

```
PAGE     1

PACK2  PACK1  DOUBLE1
-----  -----  -------
 1.05    1.1      1.0
 1.15    1.2      1.1
 1.25    1.3      1.3
 1.35    1.4      1.3
 1.45    1.5      1.4
 1.55    1.6      1.5
 1.65    1.7      1.6
 1.75    1.8      1.8
 1.85    1.9      1.8
 1.95    2.0      1.9

TOTAL
15.00   15.5     15.0
```

The PACK2 values are not rounded. They are stored and displayed as they were entered. For example, 1.15 is stored internally as:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | 00 | 00 | 11 | 5C |

The PACK1 values are rounded when stored and also when displayed. For example, the incoming value 1.15 is rounded to 1.2, and stored internally as:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 00 | 00 | 00 | 00 | 00 | 00 | 01 | 2C |

All of the DOUBLE1 values, except 1.25 and 1.75, are stored as repeating decimals in hex. For example, 1.15 is stored internally as:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 41 | 12 | 66 | 66 | 66 | 66 | 66 | 66 |

The 41 in the first byte is equivalent to 64 in hex plus the exponent. The last seven bytes are the mantissa as converted to hex by the S/390.

The DOUBLE1 values 1.25 and 1.75 are not repeating decimals internally. They are terminating decimals in hex, so they round up, when displayed, to 1.3 and 1.8 respectively. For example, 1.25 is stored internally as:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 41 | 14 | 00 | 00 | 00 | 00 | 00 | 00 |

Since the PACK1 values are rounded up before they are stored, the PACK1 total is 0.5 higher than the PACK2 total. The D field total is the same as the PACK2 total because the D field values are stored as input, and then rounded for display.

# Rounding in Calculations and Conversions

All computations are done in floating-point arithmetic. Packed fields are converted to D internally, then back to P.

When a field with decimal places is computed to an integer field, the decimal places are truncated, and the resulting value is the integer part of the input value.

When a field with decimal places is computed from one format to another, two conversions take place:

1. First the field is converted internally to floating-point notation.
2. Second, the result of this conversion is converted to the specified format. At this point, the rounding algorithm described previously is applied.

*Example* **Redefining Field Formats**

The following example illustrates some differences in the way packed fields, floating-point fields, and integer fields are stored and displayed. It also shows database values redefined to a format with a larger number of decimal places.

Master File:

```
FILE=EXAMPLE, SUFFIX=FOC
 SEGNAME=ONLY, SEGTYPE=S1,$
  FIELD=PACKED2,,P9.2,$
  FIELD=DOUBLE2,,D9.2,$
  FIELD=FLOAT2,, F9.2,$
  FIELD=INTEGER,,I9  ,$
```

Program to Load Data:

This MODIFY creates a file with four fields: a P field with two decimal places, a D field with two decimal places, an F field with two decimal places, and an integer field. The same data values are then loaded into each field.

```
CREATE FILE EXAMPLE

MODIFY FILE EXAMPLE
FIXFORM PACKED2/9 X1 DOUBLE2/9 X1 FLOAT2/9 X1 INTEGER/9
MATCH PACKED
  ON MATCH REJECT
  ON NOMATCH INCLUDE
DATA
1.6666666 1.6666666 1.6666666 1.6666666
125.16666 125.16666 125.16666 125.16666
5432.6666 5432.6666 5432.6666 5432.6666
4.1666666 4.1666666 4.1666666 4.1666666
5.5       5.5       5.5       5.5
106.66666 106.66666 106.66666 106.66666
7.2222222 7.2222222 7.2222222 7.2222222
END
```

Report Request:

A DEFINE command creates temporary fields that are equal to PACKED2, DOUBLE2, and FLOAT2, with redefined formats containing four decimal places instead of two. These DEFINE fields illustrate the differences in the way packed fields and floating-point fields are stored and displayed.

The request prints the values and a total for all four database fields, and for the three DEFINE fields.

```
DEFINE FILE EXAMPLE
PACKED4/P9.4=PACKED2;
DOUBLE4/D9.4=DOUBLE2;
FLOAT4/D9.4=FLOAT2;
END
```

```
TABLE FILE EXAMPLE
PRINT PACKED2 PACKED4 DOUBLE2 DOUBLE4 FLOAT2 FLOAT4 INTEGER
ON TABLE SUMMARIZE
END
```

The resulting report follows:

```
PAGE     1


  PACKED2   PACKED4    DOUBLE2    DOUBLE4    FLOAT2     FLOAT4   INTEGER
  -------   -------    -------    -------    ------     ------   -------
     1.67    1.6700       1.67     1.6667      1.67     1.6667         2
   125.17  125.1700     125.17   125.1667    125.17   125.1667       125
  5432.67 5432.6700   5,432.67 5,432.6666   5432.66 5,432.6641      5433
     4.17    4.1700       4.17     4.1667      4.17     4.1667         4
     5.50    5.5000       5.50     5.5000      5.50     5.5000         6
   106.67  106.6700     106.67   106.6667    106.67   106.6667       107
     7.22    7.2200       7.22     7.2222      7.22     7.2222         7

TOTAL
  5683.07 5683.0700   5,683.06 5,683.0555   5683.04 5,683.0529      5684
```

In this example, the PACKED2 sum is an accurate sum of the displayed values, which are the same as the stored values. The PACKED4 values and total are the same as the PACKED2 values.

The DOUBLE2 sum looks off by .01; it is not the sum of the printed values but a rounded sum of the stored values. The DEFINEd DOUBLE4 values show that the DOUBLE2 values are actually rounded from the stored values. The DOUBLE4 values and sum show more of the decimal places from which the DOUBLE2 values are rounded.

The FLOAT2 total seems off by .03. Like the DOUBLE2 total, the FLOAT2 total is a rounded total of the stored FLOAT2 values. F fields are not accurate beyond 8 digits, as the FLOAT4 column shows.

The integer sum is an accurate total. Like packed fields, the storage values and displayed values are the same.

# DEFINE and COMPUTE

DEFINE and COMPUTE may give different results for rounded fields. DEFINEd fields are treated like database fields, while COMPUTEd fields are calculated on the results of the display command in the TABLE request. The following example illustrates this difference:

```
DEFINE FILE EXAMPLE
DEFP3/P9.3=PACKED2/4;
END
```

```
TABLE FILE EXAMPLE
PRINT PACKED2 DEFP3
COMPUTE COMPP3/P9.3=PACKED2/4;
ON TABLE SUMMARIZE
END
```

The following report results:

```
PAGE     1


 PACKED2      DEFP3     COMPP3
 -------      -----     ------
    1.67       .417       .417
  125.17     31.292     31.292
 5432.67   1358.167   1358.167
    4.17      1.042      1.042
    5.50      1.375      1.375
  106.67     26.667     26.667
    7.22      1.805      1.805

TOTAL
 5683.07   1420.765   1420.767
```

The DEFP3 field is the result of a DEFINE. The values are treated like database field values. The printed total, 1420.765, is the sum of the printed DEFP3 values, just as the PACKED2 total is the sum of the printed PACKED2 values.

The COMPP3 field is the result of a COMPUTE. The printed total, 1420.767, is calculated from the total sum of PACKED2 (5683.07 / 4).

# Index

## $



## ?



## A



## B

## C



## D

## E



## F

## G



## H



## I

## J



## K



## L



## M

## N



## O



## P

## Q



## R

## S

## T

## U



## V



## W



## Y



## Z

# 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