Information Builders

FOCUS for Mainframe

Describing Data

Version 7.6

DN1001058.0308

Contents

Preface

This documentation describes how to use FOCUS Version 7.6 in the z/VM and z/OS environments. It is intended for all FOCUS users. This manual is part of the FOCUS documentation set.

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

The documentation set consists of the following components:

- 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 Using Functions manual describes internal functions and user-written subroutines.
- 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 z/OS environments.

The users' documentation for FOCUS Version 7.3 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.

How This Manual Is Organized

This manual includes the following chapters:

Chapter/Appendix		Contents
1	Understanding a Data Source Description	Introduces data source descriptions and explains how to use them.
2	Identifying a Data Source	Documents how to describe general aspects of a data source.
3	Describing a Group of Fields	Documents how to describe groups of related fields or segments of a data source.
4	Describing an Individual Field	Documents how to describe specific field-level information of a data source.
5	Describing a Sequential, VSAM, or ISAM Data Source	Provides supplementary information specific to sequential, VSAM, and ISAM data sources.
6	Describing a FOCUS Data Source	Provides information specific to FOCUS data sources.
7	Defining a Join in a Master File	Describes how to create a new relationship between any two segments that have at least one field in common by joining them.
9	Checking and Changing a Master File: CHECK	Describes how to use the CHECK command to validate a data source description.
10	Accessing a FOCUS Data Source: USE	Describes how you can use DBA security features to provide security for any FOCUS data source.
11	Providing Data Source Security: DBA	Describes utilities to create new FOCUS data sources and to refresh existing data sources after the structure has changed.
A	Master Files and Diagrams	Contains Master Files and diagrams of sample data sources used in the documentation examples.
B	Error Messages	Describes how to obtain information about error messages.

Chapter/Appendix		Contents
C	User Exits for a Non-FOCUS Data Source	Describes how to read non-FOCUS data sources with user-written procedures.
D	Rounding in FOCUS	Describes how FOCUS numeric fields store and display data, how rounding occurs in calculations, and what happens in conversion from one format to another.
E	File Description Attribute Summary	Lists the attributes that can be used in Master and Access Files for FOCUS and XFOCUS data sources.

Documentation Conventions

The following conventions apply throughout this manual:

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

Convention	Description
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

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

Customer Support

Do you have questions about FOCUS?

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

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

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

Information You Should Have

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

- ❏ Your six-digit site code (*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 data sources, have you tried executing a query containing just the code to access the data source?
- ❏ Do you have a trace file?
- ❏ How is the problem affecting your business? Is it halting development or production? Do you just have questions about functionality or documentation?

User Feedback

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

Thank you, in advance, for your comments.

Information Builders Consulting and Training

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

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

1 Understanding a Data Source Description

Information Builders products provide a flexible data description language, which you can use with many types of data sources, including:

- Relational, such as DB2, Oracle, Sybase, and Teradata.
- Hierarchical, such as IMS, FOCUS and XFOCUS.
- Network, such as CA-IDMS.
- Indexed, such as ISAM and VSAM.
- Sequential, both fixed-format and free-format.
- Multi-dimensional, such as Essbase.

You can also use the data description language and related facilities to:

- Join different types of data sources to create a temporary structure from which your request can read or write.
- Define a subset of fields or columns to be available to users.
- Logically rearrange a data source to access the data in a different order.

Topics:

A Note About Data Source Terminology

Different types of data sources make use of similar concepts but refer to each differently. For example, the smallest meaningful element of data is called a *field* by many hierarchical database management systems and indexed data access methods, but called a *column* by relational database management systems.

There are other cases in which a common concept is identified by a number of different terms. For simplicity, we use a single set of standardized terms. For example, we usually refer to the smallest meaningful element of data as a field, regardless of the type of data source. However, when required for clarity, we use the term specific to a given data source. Each time we introduce a new standard term, we define it and compare it to equivalent terms used with different types of data sources.

What Is a Data Source Description?

When your application accesses a data source, it needs to know how to interpret the data that it finds. Your application needs to know about:

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

To obtain the necessary information, your application reads a data source description. The primary component of a data source description is called a Master File. A Master File describes the structure of a data source and its fields. For example, it includes information such as field names and data types.

For some data sources, an Access File supplements a Master File. An Access File includes additional information that completes the description of the data source. For example, it includes the full data source name and location. You require one Master File—and, for some data sources, one Access File—to describe a data source.

How an Application Uses a Data Source Description

Master Files and Access Files are stored separately, apart from the associated data source. Your application uses a data source's Master File (and if required, the corresponding Access File) to interpret the data source in the following way:

1. Identifies, locates, and reads the Master File for the data source named in a request.

   If the Master File is already in memory, your application uses the memory image and then proceeds to locate and read the data source.

   If the Master File is not in memory, the application locates the Master File on a storage device and loads it into memory, replacing any existing Master File in memory.

   If your Master File references other data sources as cross-referenced segments, or if a JOIN command is in effect for this file, the cross-referenced Master Files are also read into memory.

2. Reads the security rules if Information Builders data source security (DBA) has been specified for the data source and ensures that user access is allowed based on any DBA security specified.

3. Locates and reads the Access File for the data source named in the request, if that data source requires an Access File.

4. Locates and reads the data source.

   The data source contents are interpreted based on the information in the Master File and, if applicable, the Access File.

What Does a Master File Describe?

In this section:

A Master File enables you to:

- ❏ Identify the name and type of a data source.
- ❏ Identify and relate groups of fields.
- ❏ Describe individual fields.

Note: Every Master File must contain at least one segment declaration and one field declaration. If a required attribute is not assigned a specific value, a comma must be used as a placeholder.

Identifying a Data Source

In order to interpret data, your application needs to know the name you are using to identify the data source and what type of data source it is. For example, is it a DB2 data source, an Oracle data source, or a FOCUS data source?

For more information, see Chapter 2, *Identifying a Data Source*.

Identifying and Relating a Group of Fields

A Master File identifies and relates groups of fields that have a one-to-one correspondence with each other—in Master File terms, a segment; in relational terms, a table.

You can join data sources of the same type (using a Master File or a JOIN command) and data sources of different types (using a JOIN command). For example, you can join two DB2 data sources to a FOCUS data source, and then to a VSAM data source.

For more information about defining and relating groups of fields, see Chapter 3, *Describing a Group of Fields*.

Describing a Field

Every field has several characteristics that you must describe in a Master File, such as type of data and length or scale. A Master File can also indicate optional field characteristics. For example, a Master File can specify if the field can have a missing value, and can provide descriptive information for the field.

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

For more information, see Chapter 4, *Describing an Individual Field*.

Creating a Data Source Description

In this section:

You can create a Master File and Access File for a data source in several ways. If the data source:

- Has an existing description—such as a native schema or catalog, or a COBOL File Description—you can use a tool to automatically generate the Master File and Access File from the existing description.
- Does not have an existing description, you can create a Master File and (if required) an Access File by coding them using Information Builders' data source description language, and specify their attributes using any text editor.

Creating a Master File and Access File Using an Editor

Create a Master File and an Access File by coding them using a text editor. You can do this in all Information Builders products. The information that you require about Master File syntax is contained in this documentation. For information about Access File syntax, see your data adapter documentation. After editing a Master File, issue the CHECK FILE command to validate the new Master File and to refresh your session's image of it.

Naming a Master File

In this section:

Master File names for FOCUS and fixed-format sequential data sources can be up to 64 characters long on z/OS, UNIX, and Windows platforms. Except where noted, this length is supported in all functional areas that reference a Master File.

Using Long Master File Names on z/OS

In the z/OS environment, file and member names are limited to eight characters. Therefore, longer Master File names are assigned eight-character names to be used when interacting with the operating system. Use the following to implement Master File names longer that eight characters:

- A LONGNAME option for the DYNAM ALLOCATE command, which creates the long Master File name and performs the allocation. This DYNAM option is described in *How to Allocate a Long Master File Name in z/OS* on page 28.
- An eight-character naming convention for member names associated with long Master File names. This convention is described in *Member Names for Long Master File Names in z/OS* on page 26.
- A long Master File attribute, $ VIRT, which contains the long name to be used when interacting with the Master File and the operating system. This attribute is described in *How to Implement a Long Master File Name in z/OS* on page 28.

Member Names for Long Master File Names in z/OS

How to:

Implement a Long Master File Name in z/OS

Allocate a Long Master File Name in z/OS

Free an Allocation for a Long Master File Name

Example:

Long Master File Names and Corresponding Member Names

Using a Long Master File Name on z/OS

Reference:

Usage Notes for Long Master File Names

The DYNAM ALLOC command with the LONGNAME option automatically creates a member for the long Master File name in the PDS allocated to ddname HOLDMAST.

The member name consists of three parts: a prefix consisting of the leftmost characters from the long name, followed by a left brace character ({), followed by an index number. This naming convention is in effect for all long Master Files allocated using DYNAM or created using the HOLD command. The length of the prefix depends on how many long names have a common set of leftmost characters:

- The first ten names that share six or more leftmost characters have a six-character prefix and a one-character index number, starting from zero.
- Starting with the eleventh long name that shares the same leftmost six characters, the prefix becomes five characters, and the index number becomes two characters, starting from 00.

This process can continue until the prefix is one character and the index number is six characters. If you delete one of these members from the HOLDMAST PDS, the member name will be reused for the next new long name created with the same prefix.

Example: Long Master File Names and Corresponding Member Names

The following table lists sample long names with the corresponding member names that would be assigned under z/OS.

Long Name	Member Name
EMPLOYEES_ACCOUNTING	EMPLOY{0
EMPLOYEES_DEVELOPMENT	EMPLOY{1
EMPLOYEES_DISTRIBUTION	EMPLOY{2
EMPLOYEES_FINANCE	EMPLOY{3
EMPLOYEES_INTERNATIONAL	EMPLOY{4
EMPLOYEES_MARKETING	EMPLOY{5
EMPLOYEES_OPERATIONS	EMPLOY{6
EMPLOYEES_PERSONNEL	EMPLOY{7
EMPLOYEES_PUBLICATIONS	EMPLOY{8
EMPLOYEES_RESEARCH	EMPLOY{9
EMPLOYEES_SALES	EMPLO{00
EMPLOYEES_SUPPORT	EMPLO{01

Syntax: How to Implement a Long Master File Name in z/OS

To relate the short name to its corresponding long name, the first line of a long Master File contains the following attribute

```
$ VIRT = long_filename
```

where:

long_filename

Is the long name, up to 64 characters.

Syntax: How to Allocate a Long Master File Name in z/OS

```
DYNAM ALLOC DD ddname LONGNAME long_filename DS physical_filename
```

where:

ddname

Is the one- to eight-character member name in a PDS allocated to DD MASTER.

long_filename

Is the long Master File name. The DYNAM command creates a copy of the short Master File in the PDS allocated to DD HOLDMAST. The member in HOLDMAST conforms to the eight character naming convention for long names. The Master File has the $ VIRT attribute on the top line, which contains the long name.

Note: The copy, not the member ddname, is the Master File used when you reference the long name in a request.

physical_filename

Is the data set name of the FOCUS or fixed-format sequential data source.

After you have allocated the long name, you can reference the data source using the long Master File name or the short ddname.

Syntax: How to Free an Allocation for a Long Master File Name

```
DYNAM FREE LONGNAME long_filename
```

where:

long_filename

Is the long Master File name.

After issuing the DYNAM FREE LONGNAME command, you cannot reference the data source using the long Master File name. However, you can reference it using the short ddname that was specified in the DYNAM ALLOC command.

Example: Using a Long Master File Name on z/OS

To reference the EMPLOYEE data source as EMPLOYEE_DATA, dynamically allocate the long name:

```
DYNAM ALLOC DD EMPLOYEE LONGNAME EMPLOYEE_DATA -
  DS USER1.EMPLOYEE.FOCUS SHR REU
```

You can now issue a request using the long name:

```
TABLE FILE EMPLOYEE_DATA
PRINT CURR_SAL
BY LAST_NAME BY FIRST_NAME
END
```

The output is:

```
LAST_NAME       FIRST_NAME         CURR_SAL
---------       ----------         --------
BANNING         JOHN             $29,710.00
BLACKWOOD       ROSEMARIE        $21,790.00
CROSS           BARBARA          $27,072.00
GREENSPAN       MARY              $9,010.00
IRVING          JOAN             $26,872.00
JONES           DIANE            $18,490.00
MCCOY           JOHN             $18,490.00
MCKNIGHT        ROGER            $16,110.00
ROMANS          ANTHONY          $21,130.00
SMITH           MARY             $13,210.00
                RICHARD           $9,510.00
STEVENS         ALFRED           $11,010.00
```

In this example, the long Master File will exist in the HOLDMAST PDS as member EMPLOY{0. The index number after the bracket depends on the number of existing long Master Files containing the same first six leftmost characters. The content of the EMPLOYEE_DATA Master File is virtually identical to the short Master File used in the allocation. The only difference is the $ VIRT keyword on line one, which contains the long name. The FILENAME parameter also contains the long name, up to 64 characters.

```
$ VIRT=EMPLOYEE_DATA
$ Created from EMPLOYEE      MASTER
FILENAME=EMPLOYEE_DATA,
SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,     ALIAS=EID, FORMAT=A9,  $
 FIELDNAME=LAST_NAME, ALIAS=LN,  FORMAT=A15, $
          .
          .
          .
```

Reference: Usage Notes for Long Master File Names

- ❏ The FOCUS Database Server (FDS) is not supported on any platform.
- ❏ The DATASET attribute is not supported in a long name Master File.
- ❏ The ACCESSFILE attribute is not supported with long name Master Files.
- ❏ An external index is not supported.
- ❏ The LONGNAME option of the DYNAM command may only be issued from within a procedure (FOCEXEC) or Remote Procedure Call (RPC). It cannot be used to preallocate long Master Files in JCL or CLISTS on z/OS.
- ❏ Long Master Files are not designed to be edited on z/OS. Each time the DYNAM command is issued with the LONGNAME attribute, it overlays the existing member in HOLDMAST. You must make any edits (such as the addition of fields or DBA attributes, or use of the REBUILD utility) to an existing short Master File.
- ❏ ? FDT and ? FILE *longfilename* will show an internal DD alias of @000000*n*, where *n* is less than or equal to the number of existing long file allocations. Use this internal DDNAME in all queries that require a valid DDNAME, such as USE commands (z/OS only).
- ❏ VM is not supported.

What Is in a Master File?

In this section:

How to:

Example:

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

- ❏ A data source declaration.
- ❏ A segment declaration for each segment within the data source.
- ❏ A field declaration for each field within a segment.

The specifications for an Access File are similar, although the details vary by type of data source. The appropriate documentation for your data adapter indicates whether you require an Access File and, if so, what the Access File attributes are.

Syntax: How to Specify a Declaration

Each declaration specifies a series of attributes in the form

```
attribute = value, attribute = value, ... ,$
```

where:

attribute

Is a Master File keyword that identifies a file, segment, or field property. You can specify any Master File attribute by its full name, its alias, or its shortest unique truncation. For example, you can use the full attribute FILENAME or the shorter form FILE.

value

Is the value of the attribute.

A comma follows each attribute assignment, and each field declaration ends with a dollar sign ($). Commas and dollar signs are optional at the end of data source and segment declarations.

Each declaration should begin on a new line. You can extend a declaration across as many lines as you want. For a given declaration you can put each attribute assignment on a separate line, combine several attributes on each line, or include the entire declaration on a single line.

- For more information on data source declarations, see Chapter 2, *Identifying a Data Source*.
- For more information on segment declarations, see Chapter 3, *Describing a Group of Fields*.
- For more information on field declarations, see Chapter 4, *Describing an Individual Field*.

Note: In a Master File, the attribute name must be in English; the attribute value can be in any supported national language.

Improving Readability

Begin each attribute assignment in any position. You can include blank spaces between the elements in a declaration. This makes it easy for you to indent segment or field declarations to make the Master File easier to read.

You can also include blank lines to separate declarations. Blank spaces and lines are not required and are ignored by the application.

Example: Improving Readability With Blank Spaces and Blank Lines

The following declarations show how to improve readability by adding blank spaces and blank lines within and between declarations:

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

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

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

Example: Improving Readability by Extending a Declaration Across Lines

The following example extends a field declaration across several lines:

```
FIELDNAME = MEMBERSHIP, ALIAS = BELONGS, USAGE = 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' ,$
```

Adding a Comment

You can add comments to any declaration by:

- ❏ Typing a comment in a declaration line after the terminating dollar sign.
- ❏ Creating an entire comment line by placing a dollar sign at the beginning of the line.

Adding a comment line terminates the previous declaration if it has not already been terminated. Everything on a line following the dollar sign is ignored.

Comments placed after a dollar sign are useful only for those who view the Master File source code.

Example: Adding a Comment in a Master File

The following example contains two comments. The first comment follows the dollar sign on the data source declaration. The second comment is on a line by itself after the data source declaration.

```
FILENAME = EMPLOYEE, SUFFIX = FOC ,$ This is the personnel data source.
$ This data source tracks employee salaries and raises.
SEGNAME = EMPINFO, SEGTYPE = S1 ,$
```

Editing and Validating a Master File

After you manually create or edit a Master File, you should issue the CHECK FILE command to validate it. CHECK FILE reads the new or revised Master File into memory and highlights any errors in your Master File so that you can correct them before reading the data source.

The CHECK FILE PICTURE command displays a diagram illustrating the structure of a data source. You can also use this command to view information in the Master File, such as names of segments and fields, and the order in which information is retrieved from the data source when you run a request against it.

For more information, see Chapter 8, *Checking and Changing a Master File: CHECK*.

2 Identifying a Data Source

In order to interpret data, your application needs to know the name you are using to identify the data source and what type of data source it is. For example, is it a DB2 data source, a Teradata data source, or a FOCUS data source?

Topics:

- ❏ Identifying a Data Source Overview
- ❏ Specifying a Data Source Name: FILENAME
- ❏ Identifying a Data Source Type: SUFFIX
- ❏ Providing Descriptive Information for a Data Source: REMARKS
- ❏ Specifying a Physical File Name: DATASET

Identifying a Data Source Overview

In a Master File, you identify the name and the type of data source in a data source declaration. A data source declaration can include the following attributes:

- FILENAME, which identifies the name of the data source.
- SUFFIX, which identifies the type of data source.
- REMARKS, which allows you to provide descriptive information about the data source for display in graphical tools such as Report Assistant and Graph Assistant.
- ACCESSFILE, which identifies the name of the optional Access File for a FOCUS data source. See Chapter 6, *Describing a FOCUS Data Source*.
- DATASET, which identifies the physical file name if your data source has a non-standard name.

You can optionally specify a sliding date window that assigns a century value to dates stored with two-digit years, using these data source attributes:

- FDEFCENT, which identifies the century.
- FYRTHRESH, which identifies the year.

For more information on the sliding window technique, see the *Developing Applications* manual.

Specifying a Data Source Name: FILENAME

How to:

Specify a Data Source Name

Example:

Specifying a Data Source Name

The FILENAME attribute specifies the name of the data source described by the Master File. This is the first attribute specified in a Master File. You can abbreviate the FILENAME attribute to FILE.

Syntax: How to Specify a Data Source Name

```
FILE[NAME] = data_source_name
```

where:

data_source_name

Is the name of the data source that the Master File describes. The name can be a maximum of eight characters.

The file name must contain at least one alphabetic character and the remaining characters can be any combination of letters, numbers, and underscores (_). On z/OS, the file name must start with an alphabetic character.

Example: Specifying a Data Source Name

The following example specifies the data source name EMPLOYEE:

```
FILENAME = EMPLOYEE
```

Identifying a Data Source Type: SUFFIX

How to:

Identify a Data Source Type

Example:

Specifying the Type for a FOCUS Data Source

Reference:

SUFFIX Values

The SUFFIX attribute identifies the type of data source you are using—for example, a DB2 data source or a FOCUS data source. Based on the value of SUFFIX, the appropriate data adapter is used to access the data source.

The SUFFIX attribute is required for most types of data sources. It is optional for a fixed-format sequential data source. However, if you refer to a fixed-format sequential data source in a JOIN command, then the SUFFIX attribute must be declared in the Master File.

You can create your own data access module for any non-standard data source that cannot be described using a standard Master File. In this case you would assign the name of the data access module to the SUFFIX attribute in the Master File.

Syntax: How to Identify a Data Source Type

```
SUFFIX = data_source_type
```

where:

data_source_type

Indicates the type of data source or the name of a customized data access module. The default value is FIX, which represents a fixed-format sequential data source.

Example: Specifying the Type for a FOCUS Data Source

The following example specifies the data source type FOC, representing a FOCUS data source which has 4K database pages:

```
SUFFIX = FOC
```

The following example specifies the data source type XFOCUS, representing an XFOCUS data source which has 16K database pages:

```
SUFFIX = XFOCUS
```

Reference: SUFFIX Values

The following table indicates the SUFFIX value for many of the supported data source types:

Data Source Type	SUFFIX Value
ADABAS	ADBSIN or ADBSINX ADABAS (OpenVMS EDA 2.x and FOCUS 6.x)
CA-Datacom/DB	DATACOM
CA-IDMS/DB	IDMSR
CA-IDMS/SQL	SQLIDMS
DB2	DB2 or SQLDS
Fixed-format sequential	FIX This value is the default. PRIVATE (for FOCSAM user exit)
FOCUS	FOC
Free-format (also known as comma-delimited) sequential	COM, COMMA, COMT
IMS	IMS
MODEL 204	M204IN
NOMAD	NMDIN
Oracle	SQLORA
SQL/DS	SQLDS
Tab-Delimited	TABT, TAB
Teradata	SQLDBC
Token-Delimited	DFIX
VSAM	VSAM PRIVATE (for FOCSAM user exit)
XFOCUS	XFOCUS
Non-standard data source	Name of the customized data access routine.

Providing Descriptive Information for a Data Source: REMARKS

How to:

Document a Data Source

Example:

Providing Descriptive Information for an Oracle Table

The optional REMARKS attribute provides descriptive information about the data source.

You can also include descriptive information as a comment following a $ symbol in the Master File. For more information, see Chapter 1, *Understanding a Data Source Description*. Master Files support REMARKS and DESCRIPTION attributes in multiple languages. For information, see *Using Multilingual Descriptions in a Master File* in Chapter 4, *Describing an Individual Field*.

Syntax: How to Document a Data Source

```
REMARKS = 'descriptive_text'
```

where:

descriptive_text

Is descriptive information about the data source. The text can be a maximum of 76 characters long and must be enclosed within single quotation marks.

The descriptive text cannot span more than one line in the Master File. If necessary, place the entire REMARKS attribute on a line by itself. For longer descriptions, break the declaration into lines with the descriptive text on a line by itself.

Example: Providing Descriptive Information for an Oracle Table

The following example shows the data source declaration for the Oracle table ORDERS. The data source declaration includes descriptive information about the table.

```
FILENAME=ORDERS, SUFFIX=SQLORA,
 REMARKS='This Oracle table tracks daily, weekly, and monthly orders.' ,$
```

Since the descriptive information would not fit on the same line as the other data source attributes, the REMARKS attribute appears on a line by itself.

Specifying a Physical File Name: DATASET

In this section:

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

User allocation and system specific behavior is as follows:

Platform	User Allocation Command
CMS	FILEDEF
TSO	DYNAM ALLOC or TSO ALLOC

Note:

- The MODIFY FIND function does not work with the DATASET attribute. To use FIND with a data source, you must explicitly allocate the data source.
- You cannot use both the ACCESSFILE attribute and the DATASET attribute in the same Master File. For information on the ACCESSFILE attribute, see Chapter 6, *Describing a FOCUS Data Source*.

DATASET Behavior in a FOCUS Data Source

How to:

Use the DATASET Attribute at the File Level

Example:

Allocating a FOCUS Data Source Using the DATASET Attribute

Allocating a Data Source for the FOCUS Database Server

You can use the DATASET attribute on the file level of a FOCUS (including XFOCUS), fixed-format sequential, or VSAM Master File. You can use the DATASET attribute on the segment level of a FOCUS Master File. For information on specifying the DATASET attribute on the segment level of a FOCUS Master File, see Chapter 6, *Describing a FOCUS Data Source*.

If the Master File's name is present in the USE list, or the user explicitly allocated the data file, the DATASET attribute is ignored.

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

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

- A user's explicit allocation overrides the DATASET attribute if you first issue the allocation command and then issue a CHECK FILE command to clear the previous DATASET allocation.
- The USE command for FOCUS data sources overrides DATASET attributes and explicit allocations.

An alternative to the DATASET attribute for allocating FOCUS data sources is an Access File. For detailed information, see Chapter 6, *Describing a FOCUS Data Source*.

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

Syntax: How to Use the DATASET Attribute at the File Level

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

where:

filename

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

sinkname

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

In z/OS, the syntax is:

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

or

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

In CMS, the syntax is:

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

Example: Allocating a FOCUS Data Source Using the DATASET Attribute

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

For z/OS,

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

For CMS,

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

Example: Allocating a Data Source for the FOCUS Database Server

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

For z/OS,

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

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

For CMS,

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

DATASET Behavior in a Fixed-Format Sequential Data Source

How to:

Use the DATASET Attribute With a Fixed-Format Data Source

Example:

Allocating a Fixed-Format Data Source Using the DATASET Attribute

The DATASET attribute for a fixed-format sequential file can be used only at the file declaration level of the Master File and cannot contain ON *sink*. If the DATASET attribute contains ON *sink*, a message is issued and the operation is terminated.

An explicit allocation of data for this Master File is checked for when the DATASET attribute is detected. If an explicit allocation exists, the DATASET attribute is ignored. If this Master File name is not allocated, an internal command is issued to perform the allocation. This allocation is stored temporarily and is released when a new Master File is used or when the session terminates.

Syntax: How to Use the DATASET Attribute With a Fixed-Format Data Source

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

where:

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

The DATASET attribute in the Master File has the lowest priority: a user's explicit allocation overrides DATASET attributes.

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

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

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

For CMS:

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

For z/OS:

```
FILE=XX,  SUFFIX=FIX,  DATASET='USER1.SEQFILE1'
   .
   .
   .
```

DATASET Behavior in a VSAM Data Source

How to:

Use the DATASET Attribute With a VSAM Data Source

Example:

Allocating a VSAM Data Source Using the DATASET Attribute

The DATASET attribute for a VSAM data source can be used on the file declaration level of the Master File and cannot contain ON *sink*. If the DATASET attribute contains ON *sink*, a message is issued and the operation is terminated.

An explicit allocation of data for this Master File is checked for when the DATASET attribute is detected. If an explicit allocation is found, the DATASET attribute is ignored. If this Master File name is not allocated, an internal command is issued to perform the allocation. This allocation is stored temporarily and is released when a new Master File is used or when the session terminates.

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

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

Syntax: How to Use the DATASET Attribute With a VSAM Data Source

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

where:

filename

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

The DATASET attribute in the Master File has the lowest priority: a user's explicit allocation overrides DATASET attributes.

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

Example: Allocating a VSAM Data Source Using the DATASET Attribute

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

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

3 Describing a Group of Fields

Certain fields in a data source may have a one-to-one correspondence. The different groups formed can be related to one another. For some types of data sources you can define a subset, a logical view of a group. You can identify these groups and the relationships between them by using attributes in the Master and Access File, 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 50. Relating these groups to each other is described in *Relating Multiple Groups of Fields* on page 56.

This section describes the relationships of fields and how to implement them using Master File attributes. If your type of data source also requires an Access File, see the appropriate data adapter documentation for supplementary information about defining groups and group relations in the Access File.

Topics:

Defining a Single Group of Fields

In this section:

Certain fields in a data source may have a one-to-one correspondence. For each value of a field, the other fields may have exactly one corresponding value. For example, consider the EMPLOYEE data source:

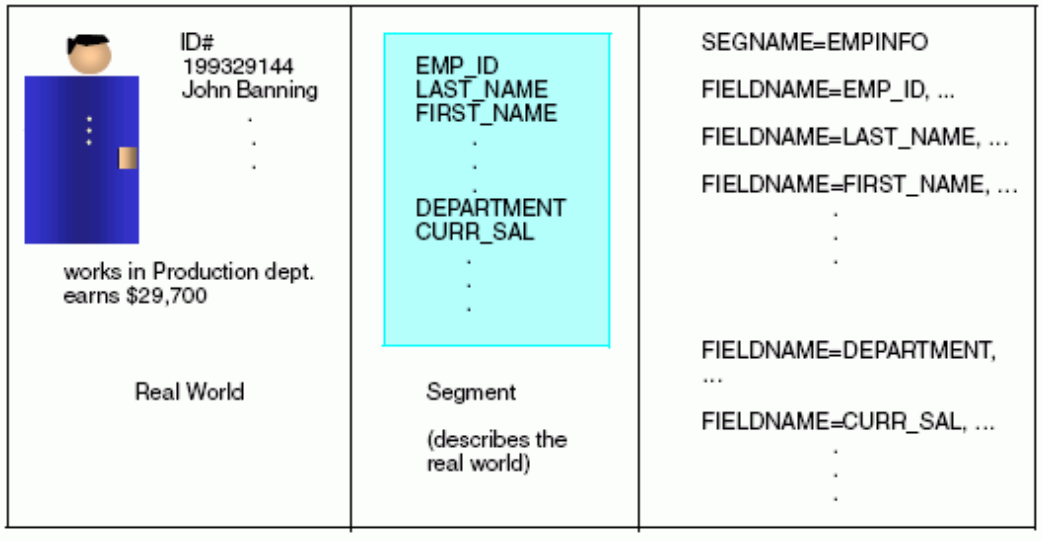
- ❏ Each employee has one ID number which is unique to that employee.
- ❏ For each ID number there is one first and last name, one date hired, one department, and one current salary.

In the data source, one field represents each of these employee characteristics. The entire group of fields represents the employee. In Master File terms, this group is called a *segment*.

Understanding a Segment

A segment is a group of fields that have a one-to-one correspondence with each other and usually describe a group of related characteristics. In a relational data source, 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, as discussed in *Relating Multiple Groups of Fields* on page 56.

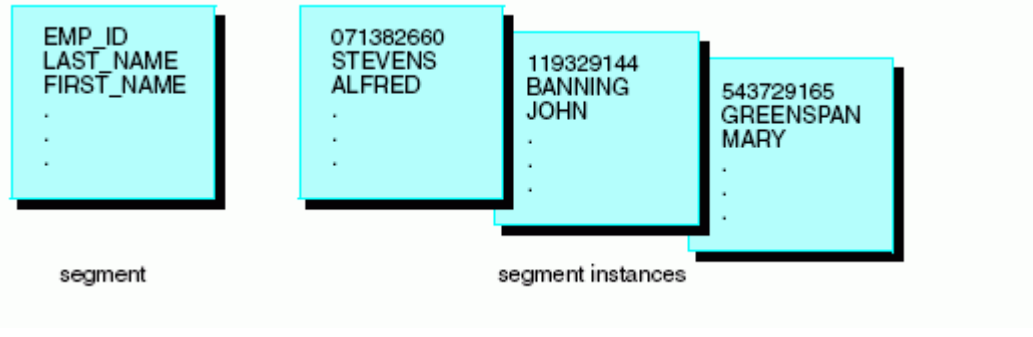
The following diagram illustrates the concept of a segment.



Understanding a Segment Instance

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

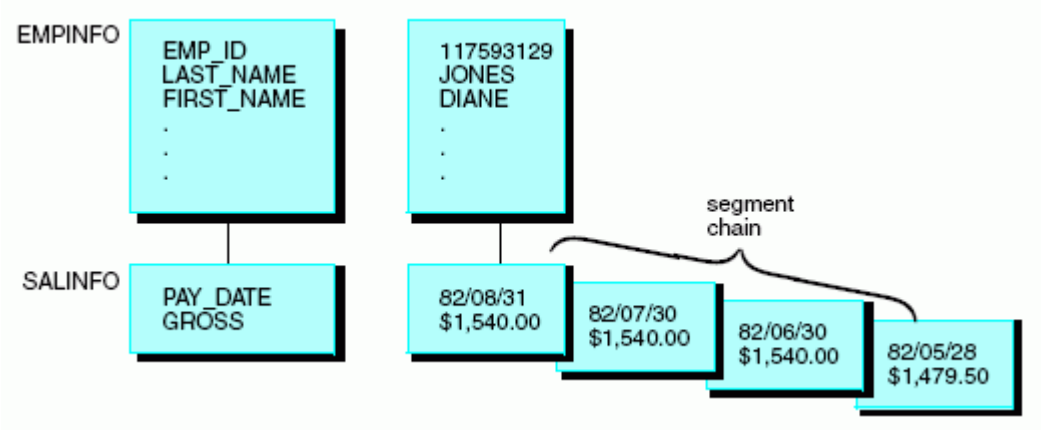
The relationship of a segment to its instances is illustrated in the following diagram.



Understanding a Segment Chain

The instances of a segment that are descended from a single parent instance are collectively known as a segment chain. In the special case of a root segment, which 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 60.

The following diagram illustrates the concept of a segment chain.



Describe a segment using the SEGNAME and SEGTYPE attributes in the Master File. The SEGNAME attribute is described in *Identifying a Segment: SEGNAME* on page 53.

Identifying a Key Field

Most segments also have key fields, which are one or more fields that uniquely identify each segment instance. In the EMPLOYEE data source, 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 data source 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 data source, the fields constituting the primary key in the Master File should be the first fields described for that segment, meaning that the field declarations should come before any others in that segment.

For FOCUS data sources, identify key fields and their sorting order using the SEGTYPE attribute in the Master File, as shown in Chapter 6, *Describing a FOCUS Data Source*. Position the key fields as the first fields in the segment.

Identifying a Segment: SEGNAME

The SEGNAME attribute identifies the segment. It is the first attribute you specify in a segment declaration. Its alias is SEGMENT.

For a FOCUS data source, the segment name may consist of up to eight characters. Segment names for an XFOCUS data source may have up to 64 characters. You can use segment names of up to 64 characters for non-FOCUS data sources, if supported by the DBMS. To make the Master File self-documenting, set SEGNAME to something meaningful to the user or the native file manager. For example, if you are describing a DB2 table, assign the table name (or an abbreviation) to SEGNAME.

In a Master File, each segment name must be unique. The only exception to this rule is in a FOCUS or XFOCUS data source, 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 their names are identical, you can still refer to them uniquely by using the CRSEGNAME attribute. See Chapter 7, *Defining a Join in a Master File*.

In a FOCUS or XFOCUS data source, you cannot change the value of SEGNAME after data has been entered into the data source. For all other types of data sources, you can change SEGNAME as long as you also change all references to it; for example, any references in the Master and Access File.

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

```
{SEGNAME|SEGMENT} = segment_name
```

where:

segment_name

Is the name that identifies this segment. For FOCUS data sources, it can be a maximum of eight characters long. Segment names for an XFOCUS data source can consist of up to 64 characters. You can use segment names of up to 64 characters for non-FOCUS data sources, if supported by the DBMS.

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

Example: Identifying a Segment

If a segment corresponds to a relational table named TICKETS, and you want to give the segment the same name, use the SEGNAME attribute in the following way:

```
SEGNAME = TICKETS
```

Identifying a Logical View: Redefining a Segment

Example:

Omitting a Field: Creating a Segment Subset

Redefining a Field: Creating a Filler Field

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

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 which includes only a subset of the segment's fields (similar to a relational view), or 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.

Use these methods with the following types of data sources:

- **Relational data sources.** Omit unwanted fields from the segment description in the Master File.
- **Non-relational data sources.** Define unwanted fields as one or more filler fields.

To restrict access explicitly at the file, segment, or field level based on user ID, field values, and other characteristics, use the DBA facility as described in Chapter 10, *Providing Data Source Security: DBA*.

Example: Omitting a Field: Creating a Segment Subset

Define a logical view for a relational data source by omitting the unwanted fields from the segment's description in the Master File. Consider the following Master File for an Oracle table named EMPFACTS:

```
FILENAME = EMPFACTS, SUFFIX = SQLORA ,$
 SEGNAME = EMPFACTS, SEGTYPE = S0 ,$
  FIELDNAME = EMP_NUMBER, ALIAS = ENUM,  USAGE = A9,     ACTUAL = A9   ,$
  FIELDNAME = LAST_NAME,  ALIAS = LNAME, USAGE = A15,    ACTUAL = A15  ,$
  FIELDNAME = FIRST_NAME, ALIAS = FNAME, USAGE = A10,    ACTUAL = A10  ,$
  FIELDNAME = HIRE_DATE,  ALIAS = HDT,   USAGE = I6YMD,  ACTUAL = DATE ,$
  FIELDNAME = DEPARTMENT, ALIAS = DPT,   USAGE = A10,    ACTUAL = A10  ,$
  FIELDNAME = SALARY,     ALIAS = SAL,   USAGE = D12.2M, ACTUAL = D8   ,$
  FIELDNAME = JOBCODE,    ALIAS = JCD,   USAGE = A3,     ACTUAL = A3   ,$
  FIELDNAME = OFFICE_NUM, ALIAS = OFN,   USAGE = 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 can code an alternative Master File that names only the desired fields:

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

Example: Redefining a Field: Creating a Filler Field

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

Consider the EMPINFO segment of the EMPLOYEE data source:

```
SEGNAME = EMPINFO, SEGTYPE = S1 ,$
  FIELDNAME = EMP_ID,       ALIAS = EID,  USAGE = A9     ,$
  FIELDNAME = LAST_NAME,    ALIAS = LN,   USAGE = A15    ,$
  FIELDNAME = FIRST_NAME,   ALIAS = FN,   USAGE = A10    ,$
  FIELDNAME = HIRE_DATE,    ALIAS = HDT,  USAGE = I6YMD  ,$
  FIELDNAME = DEPARTMENT,   ALIAS = DPT,  USAGE = A10    ,$
  FIELDNAME = CURR_SAL,     ALIAS = CSAL, USAGE = D12.2M ,$
  FIELDNAME = CURR_JOBCODE, ALIAS = CJC,  USAGE = A3     ,$
  FIELDNAME = ED_HRS,       ALIAS = OJT,  USAGE = 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 can code an alternative Master File that explicitly names only the desired fields:

```
SEGNAME = EMPINFO, SEGTYPE = S1 ,$
  FIELDNAME = EMP_ID,     ALIAS = EID, USAGE = A9  ,$
  FIELDNAME = LAST_NAME,  ALIAS = LN,  USAGE = A15 ,$
  FIELDNAME = FIRST_NAME, ALIAS = FN,  USAGE = A10 ,$
  FIELDNAME =,            ALIAS =,     USAGE = A29 ,$
```

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

In this section:

How to:

Example:

After you have described a segment, you can relate segments 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 source, you can describe the relationship.
- **Describe logical relationships.** 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 they are treated as if they were part of a single structure. The new structure can include segments from the same or different types of data sources.

  If you are creating a new FOCUS data source, you can implement segment relationships in several ways, depending upon your design goals, as described in Chapter 6, *Describing a FOCUS Data Source*.

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 should be aware of the following:

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

Facilities for Specifying a Segment Relationship

There are 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.

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

Identifying a Parent Segment: PARENT

The PARENT attribute identifies a segment's parent. Specify 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.

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

If the parent-child relationship is permanently implemented within the structure of the data source, such as within a FOCUS data source, 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 it is when you join several relational tables in the Master File, then you can change the PARENT attribute.

When you specify a parent attribute for any segment, all subsequent segments will take that parent.

Syntax: How to Identify the Parent Segment

```
PARENT = segment_name
```

where:

segment_name

If no PARENT attribute is specified in a Master File, then, by default, each segment takes the preceding segment in the Master File as its parent. If a PARENT attribute is specified, then all segments that follow will take that segment as their parent, unless explicitly specified.

It is recommended that you use the PARENT attribute for unique segments with a SEGTYPE of U.

Example: Identifying a Parent Segment

In the EMPLOYEE data source, DEDUCT's parent is SALINFO, so the segment declaration for DEDUCT includes the following attribute:

```
PARENT = SALINFO
```

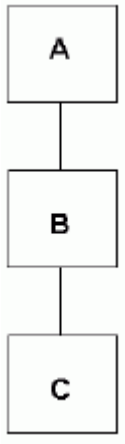
Identifying the Type of Relationship: SEGTYPE

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 differently in various types of data sources. For sequential, VSAM, and ISAM data sources, see Chapter 5, *Describing a Sequential, VSAM, or ISAM Data Source*. For FOCUS data sources, see Chapter 6, *Describing a FOCUS Data Source*. For other types of data sources, see the appropriate data adapter documentation for details.

Understanding the Efficiency of the Minimum Referenced Subtree

In any database structure consisting of more than a single table or segment, FOCUS handles retrieval by only accessing data from the minimum referenced subtree, which is a subset of the full database structure. A minimum referenced subtree consists of only those segments containing referenced fields, and any intervening segments needed to make a complete structure.

Consider the following database structure consisting of three segments, A, B, and C, with A being the parent of B, and B the parent of C. Segment A is also known as the root segment. This structure may be three different joined tables, or a single, multi-segment structure.



If a database request references fields only contained in segment A, then data in segment A only is retrieved. Likewise, if fields from segments A and B are requested, segments A and B only are retrieved. No additional retrieval costs are incurred, as would occur if all three segments were retrieved for each request.

For joined structures, there is an implicit reference to the root segment, which is always retrieved in a database request. If a request involving a joined structure references fields from segment B only, both segments A and B are retrieved since the root segment (A) is implied to link segment B. Additionally, if fields from segment C only are referenced, all three segments are retrieved since segments A and B are implied to link segment C. The retrieval costs are higher when intervening segments are retrieved for a request.

For multi-segment structures, which are defined in the same Master file, there is no implied reference to the root segment. If a request involving this type of structure references fields from one segment only, such as segment C, then one segment only, segment C, is retrieved. However, if fields from segments A and C are referenced, then all three segments are retrieved since segment B is an intervening segment required to make a complete structure. When all possible database relations are described in a single Master file, you can eliminate the costs associated with retrieving non-referenced segments.

Logical Dependence: The Parent-Child Relationship

In this section:

Logical dependence between segments is expressed in terms of the parent-child relationship: a child segment is dependent upon its parent segment. 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 the data source is accessed.

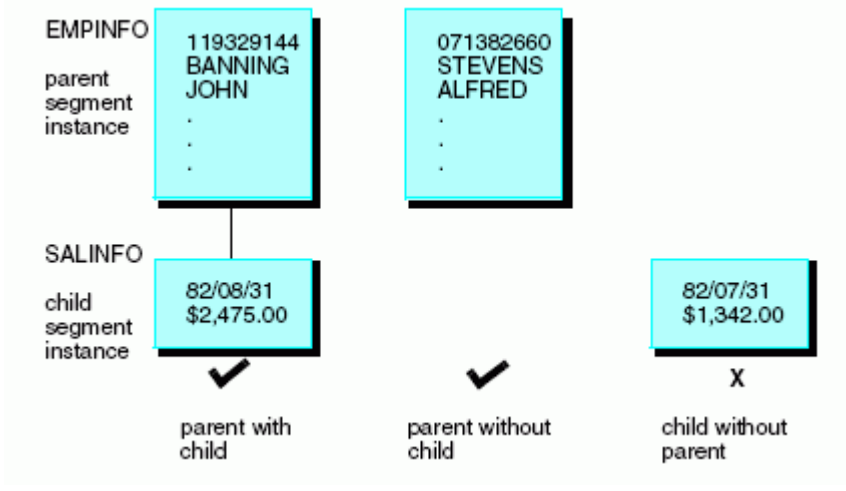
Note that if the parent-child relationship is logical and not physical, as in the case of a join, it is possible to have a child instance without a related parent instance. In this case, the child instance is not accessible through the join, although it is still accessible independently.

If a join relates the parent-child segments, the parent is known as the host segment, and the child is 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.

A Simple Parent-Child Relationship

In the EMPLOYEE data source, 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. It is possible to have in this relationship an employee identified by ID and name for whom no salary information has been entered—that is, the parent instance without the child instance. However, it is meaningless to have salary information for an employee if one does not know who the employee is—that is, a child instance without the parent instance.

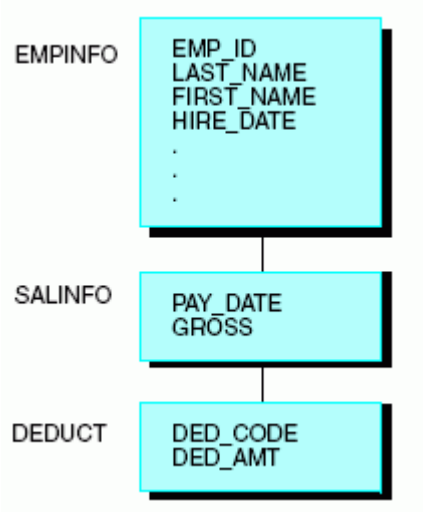
The following diagram illustrates the concept of a parent-child relationship.



A Parent-Child Relationship With Multiple Segments

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

The following diagram illustrates the concept of a parent-child relationship with multiple segments.

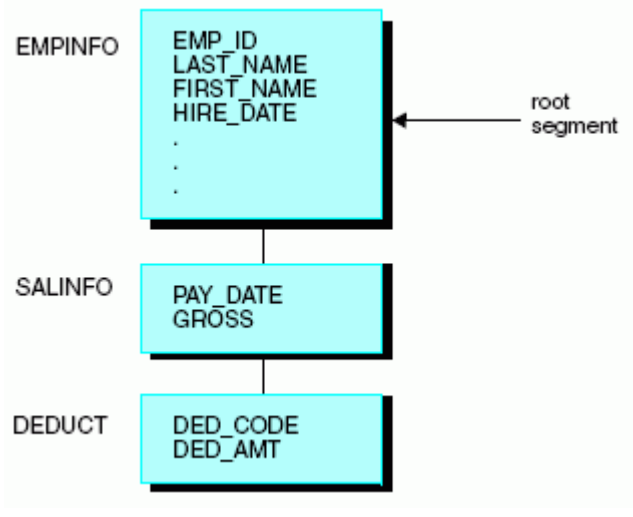


EMPINFO is related to SALINFO, and in this relationship EMPINFO is the parent segment and SALINFO is the child segment. SALINFO is also related to DEDUCT. In this second relationship, SALINFO is the parent segment and DEDUCT is the child segment. Just as SALINFO is dependent upon EMPINFO, DEDUCT is dependent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

Understanding a Root Segment

The segment that has logical precedence over the entire data structure—the parent of the entire structure—is called the *root* segment. This is 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.

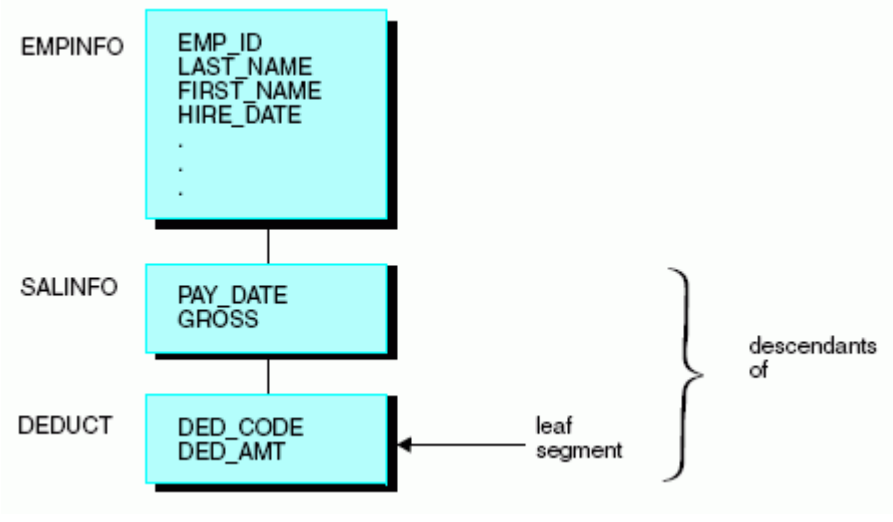
In the following diagram, EMPINFO is the root; it has no parent, and all other segments in the structure are its children, directly (SALINFO) or indirectly (DEDUCT).



Understanding a Descendant Segment

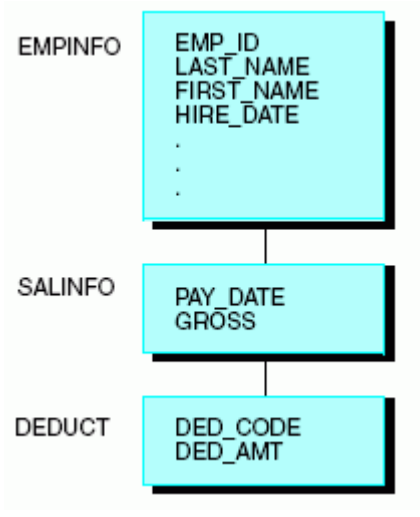
A segment's direct and indirect children are collectively known as its descendant segments. SALINFO and DEDUCT are descendants of EMPINFO. DEDUCT is also a descendant of SALINFO. A descendant segment with no children is called a *leaf* segment, because the branching of the data structure tree ends with the leaf. DEDUCT is a leaf.

The following diagram illustrates the concept of a descendant segment.



Understanding an Ancestral Segment

A segment's direct and indirect parents are its ancestral segments. In the following diagram, SALINFO and EMPINFO are ancestors of DEDUCT.



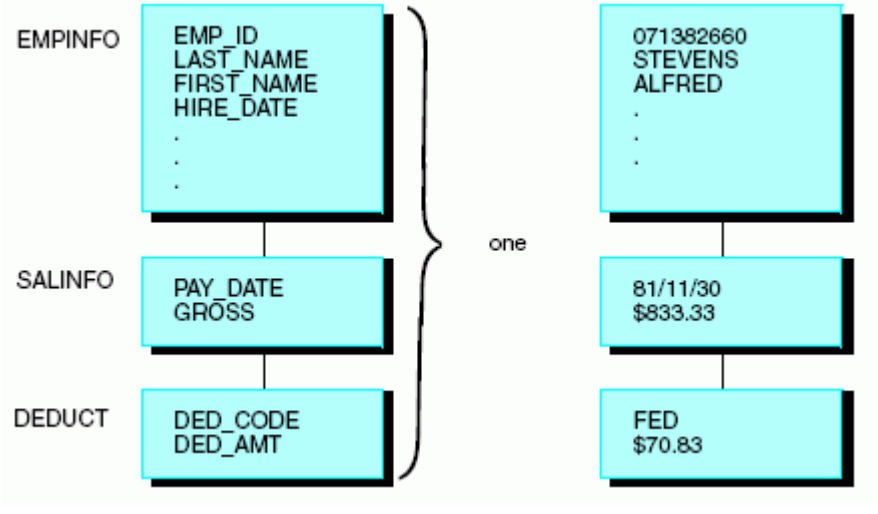
Logical Independence: Multiple Paths

In this section:

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

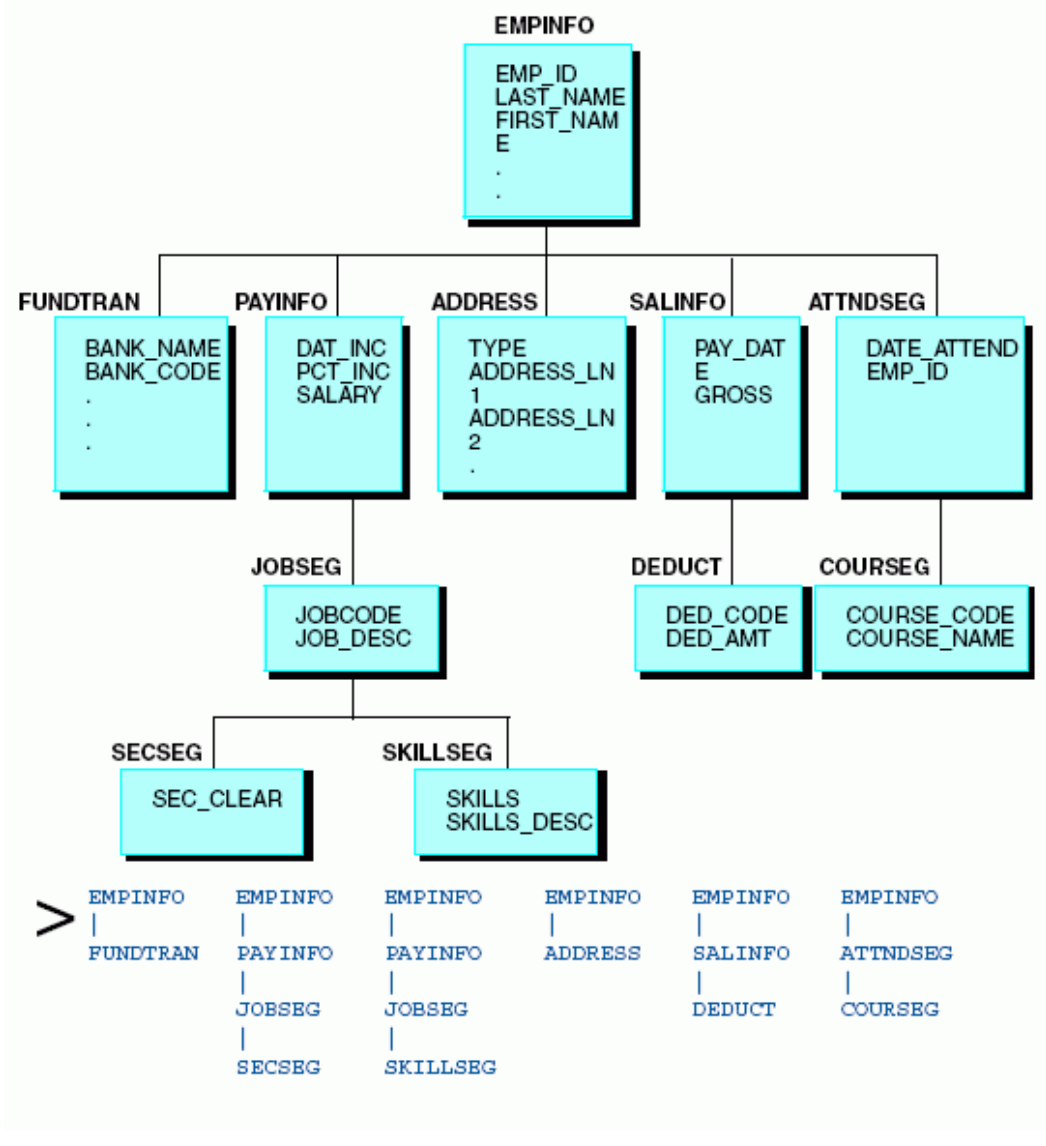
In the following view of the EMPLOYEE data source, EMPINFO, SALINFO, and DEDUCT form a path. An instance of DEDUCT (paycheck deductions) can exist only if a related instance of SALINFO (the paycheck) exists, and the instance of SALINFO can exist only if a related instance of EMPINFO (the employee) exists.



Understanding Multiple Paths

Consider the full EMPLOYEE structure, which includes the EMPLOYEE data source and the JOBFILE and EDUCFILE data sources 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 is the end of a separate path. The following diagram illustrates the concept of a multi-path data structure.



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

Cardinal Relationships Between Segments

The following types of cardinal relationships between groups of data are supported:

- 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, you can define the relationship between an Oracle table and a FOCUS data source. Note that you can join different types of data sources only by using the JOIN command, not by defining the join in the Master File or Access File.
- If you are using a network data source, you can also “rotate” the data source 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.

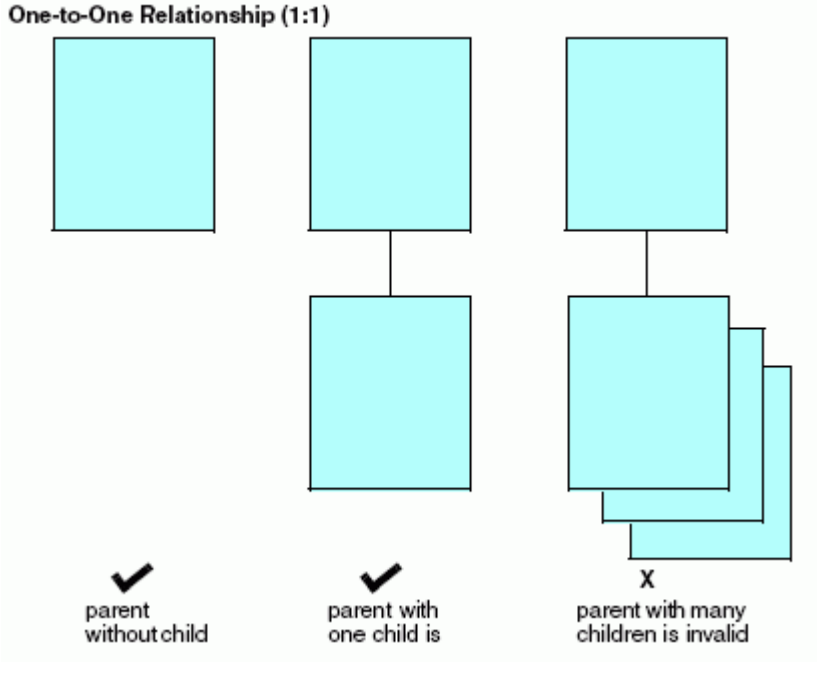
One-to-One Relationship

In this section:

Example:

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 the relationship is one-to-one, the parent instance is never related to more than one instance of the child. However, not every parent instance must have a matching child instance.

The child in a one-to-one relationship is referred to as a unique segment, because there can never be more than a single child instance. The following diagram illustrates the concept of a one-to-one relationship.

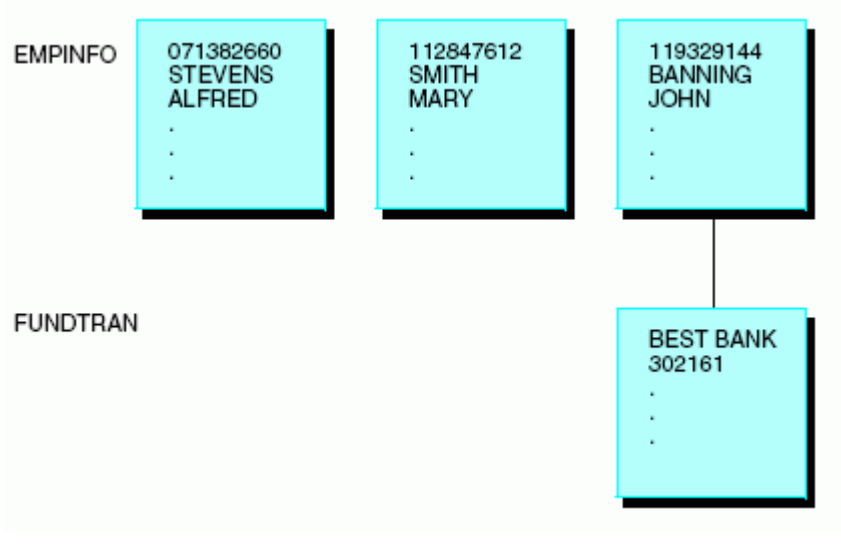


Example: Understanding a One-to-One Relationship

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

Because only one set of bank information is required 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 data source was designed with storage efficiency in mind, and so the bank fields have been put into a separate segment called FUNDTRAN. Space is only used for the banking information, creating an instance of FUNDTRAN, if it is needed. However, if banking fields are used in the parent segment (EMPINFO), the EMPINFO segment for each employee reserves space for the banking fields, even though those fields are empty in most cases. This concept is illustrated in the following diagram.



Where to Use a One-to-One Relationship

When you retrieve data, you can specify a segment as unique in order to enforce a one-to-one relationship.

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

Implementing a One-to-One Relationship in a Relational Data Source

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 data adapter documentation. Alternatively, you can join the tables by issuing the JOIN command without the ALL (or MULTIPLE) option (or with the UNIQUE option) and turning off the SQL Optimization facility with the SET OPTIMIZATION command.

Implementing a One-to-One Relationship in a Sequential Data Source

Specify this relationship between two records by issuing the JOIN command without the ALL (or MULTIPLE) option (or with the UNIQUE option).

Implementing a One-to-One Relationship in a FOCUS Data Source

Describe this relationship by specifying a SEGTYPE of U for the child segment. Alternately, you can join segments by issuing the JOIN command without the ALL (or MULTIPLE) option (or with the UNIQUE option), 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 6, *Describing a FOCUS Data Source*.

You can also describe a one-to-one segment relationship as a one-to many relationship, in the Master File or by using the JOIN command. 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.

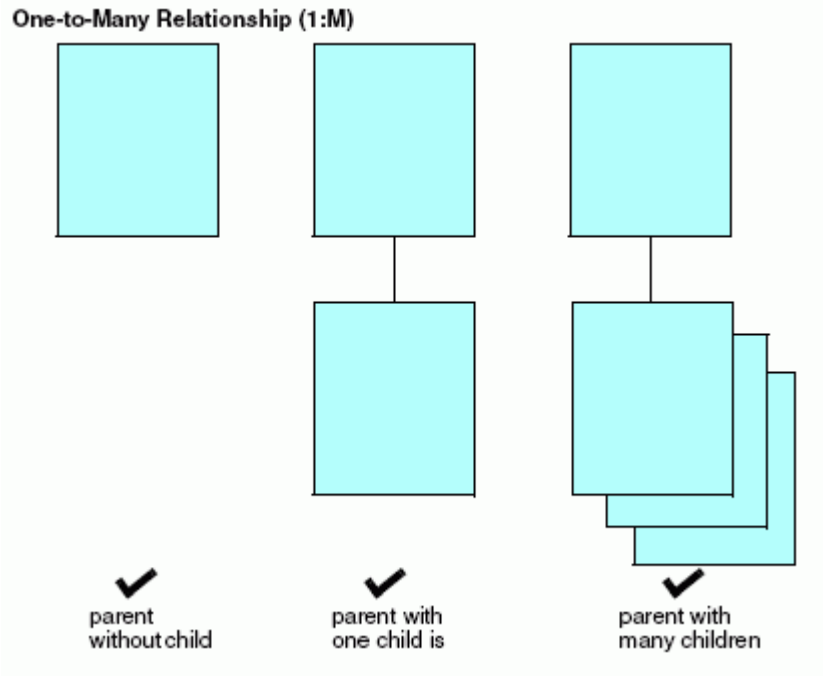
One-to-Many Relationship

In this section:

Example:

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. However, not every parent instance needs to have matching child instances.

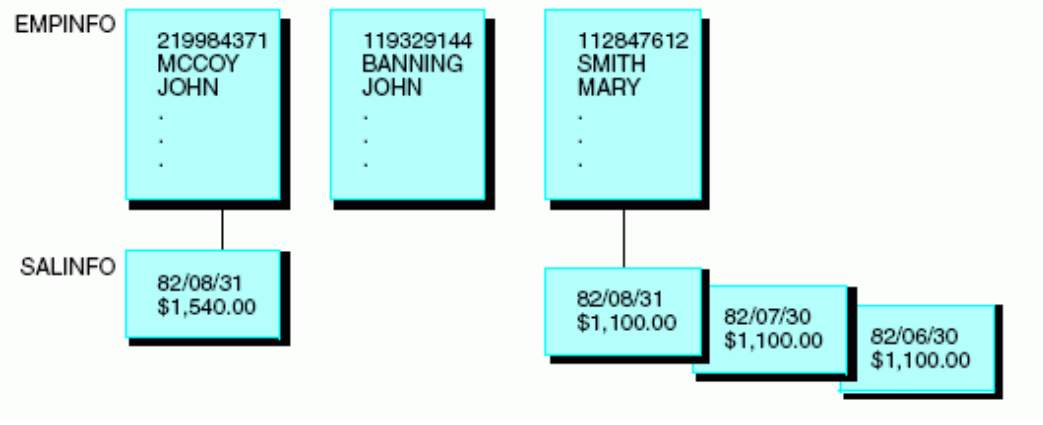
The following diagram illustrates the concept of a one-to-many relationship.



Example: Understanding a One-to-Many Relationship

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

The following diagram further illustrates the concept of a one-to-many relationship.



Implementing a One-to-Many Relationship in a Relational Data Source

Describe this relationship by joining the tables in the Master File and specifying a SEGTYPE of SO for the child table. For more information on joining the tables in a Master File, see the appropriate data adapter documentation. Alternately, you can join the tables by issuing the JOIN command with the ALL or MULTIPLE option.

Implementing a One-to-Many Relationship in a VSAM or Sequential Data Source

You can describe a one-to-many relationship between a record and a group of multiply occurring fields within the record.

- The OCCURS attribute specifies how many times the field (or fields) occur.
- The POSITION attribute specifies where in the record the field (or fields) occur if they are not at the end of the record.
- The ORDER field determines the sequence number of an occurrence of a field.
- The PARENT attribute indicates the relationship between the singly and multiply occurring fields.

The OCCURS and POSITION attributes and the ORDER field are all described in Chapter 5, *Describing a Sequential, VSAM, or ISAM Data Source*.

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 a Sequential, VSAM, or ISAM Data Source*.

You can also specify a one-to-many relationship between two records in different data sources by issuing the JOIN command with the ALL or MULTIPLE option, or defining the join in the Master File. See the *Creating Reports* manual for information about the JOIN command, and see Chapter 7, *Defining a Join in a Master File*, for information about joins in a Master File.

Implementing a One-to-Many Relationship in a FOCUS Data Source

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 or MULTIPLE option 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 6, *Describing a FOCUS Data Source*.

Many-to-Many Relationship

In this section:

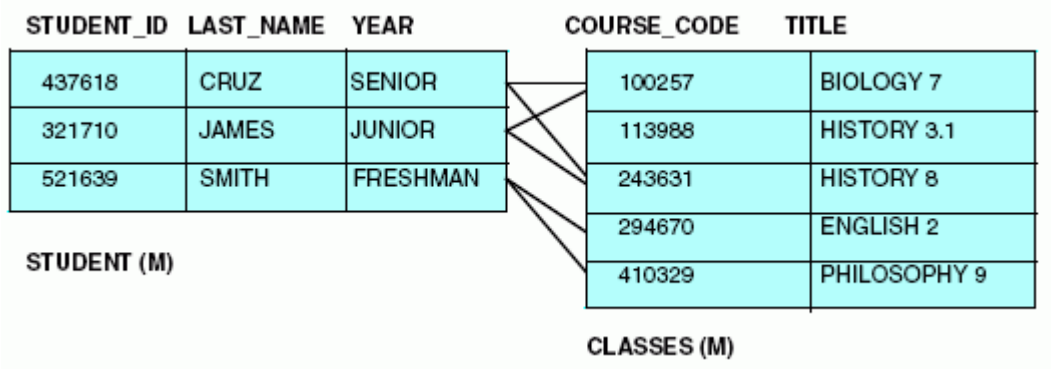
Example:

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.

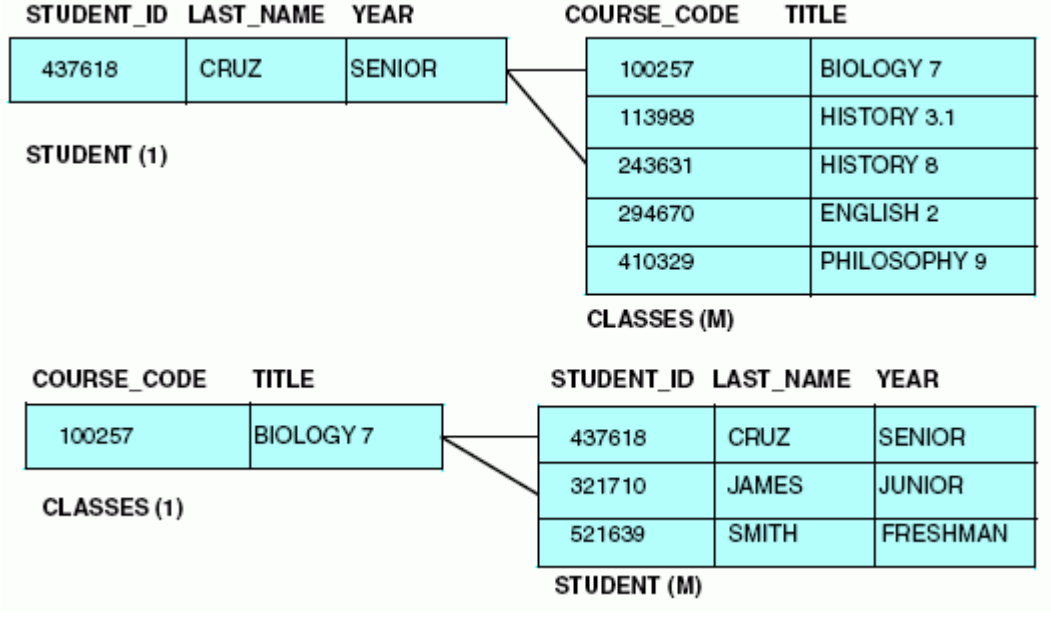
Implementing a Many-to-Many Relationship Directly

A direct many-to-many relationship can exist between two relational tables. The STUDENT table contains one row for each student enrolled at a college, and the CLASSES table contains one row for each class offered at the college. Each student can take many classes, and many students can take each class.

The many-to-many type of relationship 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). This type of relationship is illustrated in the following diagram.



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

Example: Implementing a Many-to-Many Relationship Directly

You can 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
```

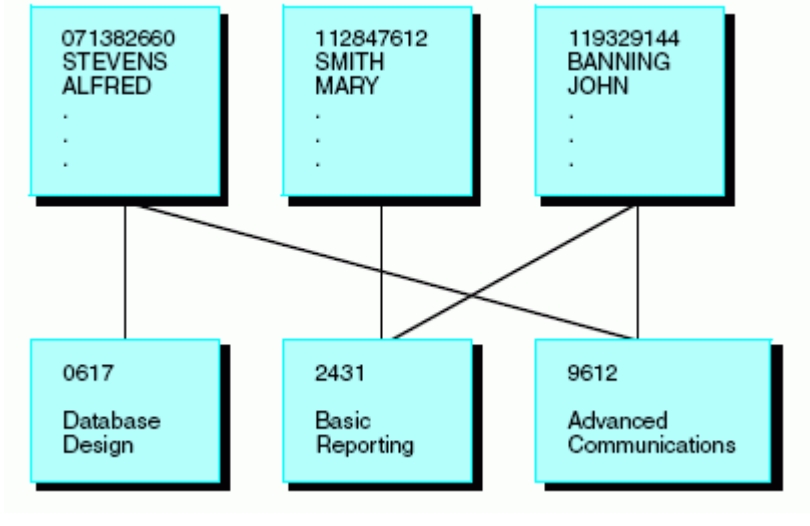
You can describe the relationship from the perspective of the CLASSES table as follows:

```
JOIN COURSE_CODE IN CLASSES TO ALL COURSE_CODE IN STUDENT
```

Implementing a Many-to-Many Relationship Indirectly

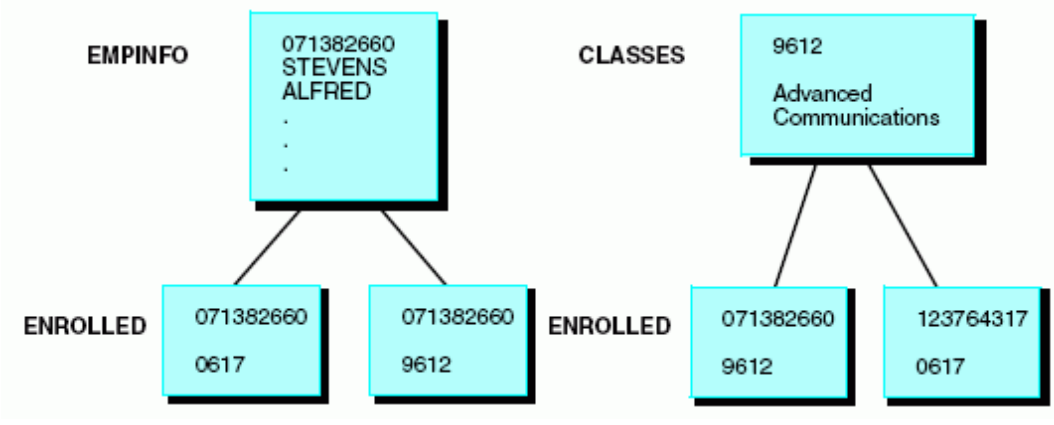
Some non-relational data sources cannot represent a many-to-many relationship directly. However, they can represent it indirectly, and you can describe it as such.

Consider the EMPINFO segment in the EMPLOYEE data source and the CLASSES segment in a hypothetical SCHOOL data source. Each instance of EMPINFO describes one employee, and each instance of CLASSES describes one course. Each employee can take many courses, and many employees can take each course, so this is a many-to-many relationship. This type of relationship is illustrated in the following diagram.



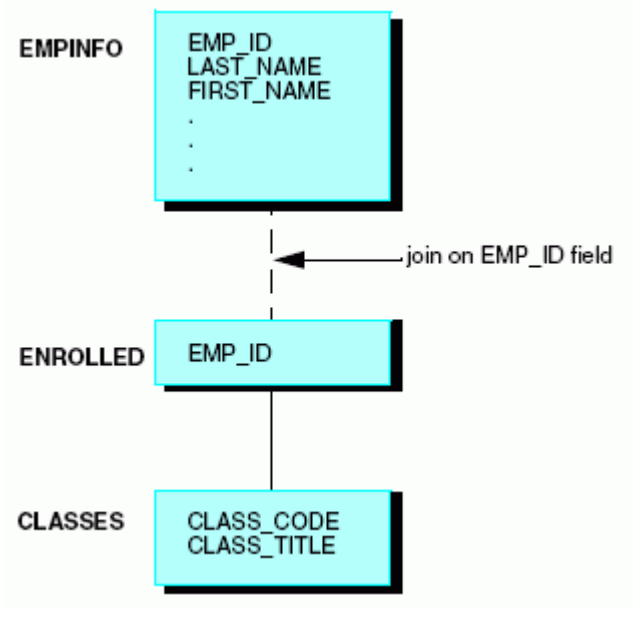
However, because some types of data sources cannot represent such a relationship directly, you must introduce a mediating segment called ENROLLED. This new segment contains the keys from both of the original segments, EMP_ID and CLASS_CODE, representing the relationship between the two original segments.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

These relationships are illustrated in the following diagram.



The next step is to make the mediating segment a child of one of the two original segments. You can design the SCHOOL data source 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. Yet as part of the SCHOOL data source, 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.

This type of join is illustrated in the following diagram.



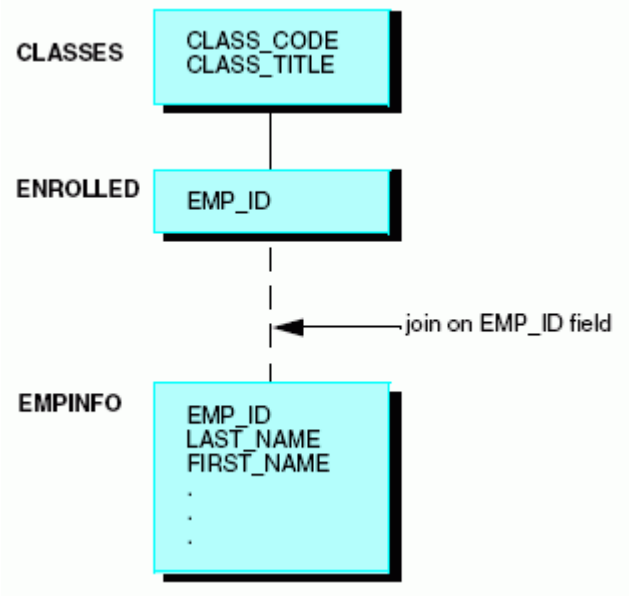
When the original M:M relationship is seen from this perspective, it looks like a 1:M:1 relationship. That is, one employee (EMPINFO) is enrolled many times (ENROLLED), and each enrollment is for a single class (CLASSES).

When you report from or update the new structure at any one time, the relationship is seen from the perspective of one of the original segments—in this case, from EMPINFO or CLASSES. Determine which segment's perspective is used by making that segment the parent in the join. Describe the join using the JOIN command, or for FOCUS data sources, in the Master File. If you make the mediating segment, in this case ENROLLED, the child (cross-referenced) segment in the join, you implement the relationship as a standard one-to-many; if you make it the parent (host) segment, you implement the relationship as a standard one-to-one join.

For example, you can use the JOIN command to describe the relationship from the perspective of the CLASSES segment, making ENROLLED the join's host:

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

The new structure is illustrated in the following diagram.



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

Recursive Relationships

Example:

A Recursive Join With a Single Segment

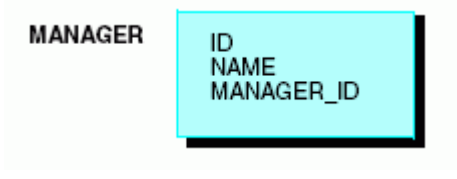
A Recursive Join With Multiple Segments

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, or even the same segment, to itself. This technique is called a recursive join.

See the *Creating Reports* manual for more information on recursive joins.

Example: A Recursive Join With a Single Segment

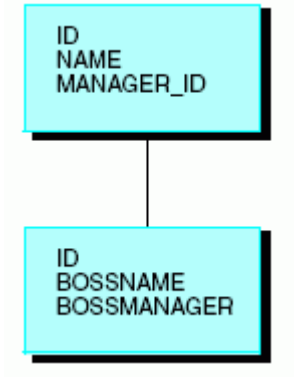
Assume that you have a single-segment data source called MANAGER, which includes the ID number of an employee, the employee's name, and the ID number of the employee's manager, as shown in the following image.



If you want to generate a report showing every employee's ID number and name, and every manager's ID number and name, you must join the segment to itself. Issue the following command:

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

This creates the following structure:



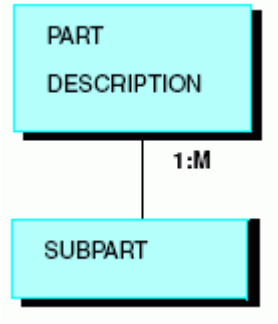
Note: You can refer to fields uniquely in cross-referenced recursive segments by prefixing them with the first four letters of the join name (BOSS in this example). The only exception is the cross-referenced field, for which the alias is prefixed instead of the field name.

After you have issued the join, you can generate an answer that looks like this:

```
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
```

Example: A Recursive Join With Multiple Segments

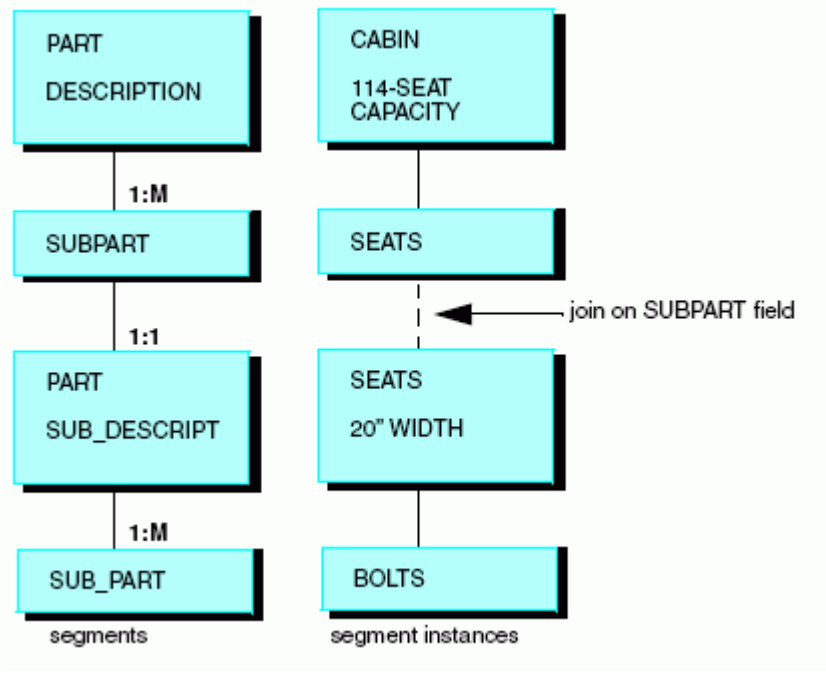
You can join larger structures recursively as well. For example, consider a two-segment data source called AIRCRAFT that stores a bill-of-materials for an aircraft company. The root segment has the name and description of a part, and the child segment has the name of a subpart. For each part, there can be many subparts. This type of joined structure is illustrated in the following diagram.



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. It is redundant to give each occurrence of a subpart its own segment instance. Instead, use the two-segment design shown previously and then join the data source to itself:

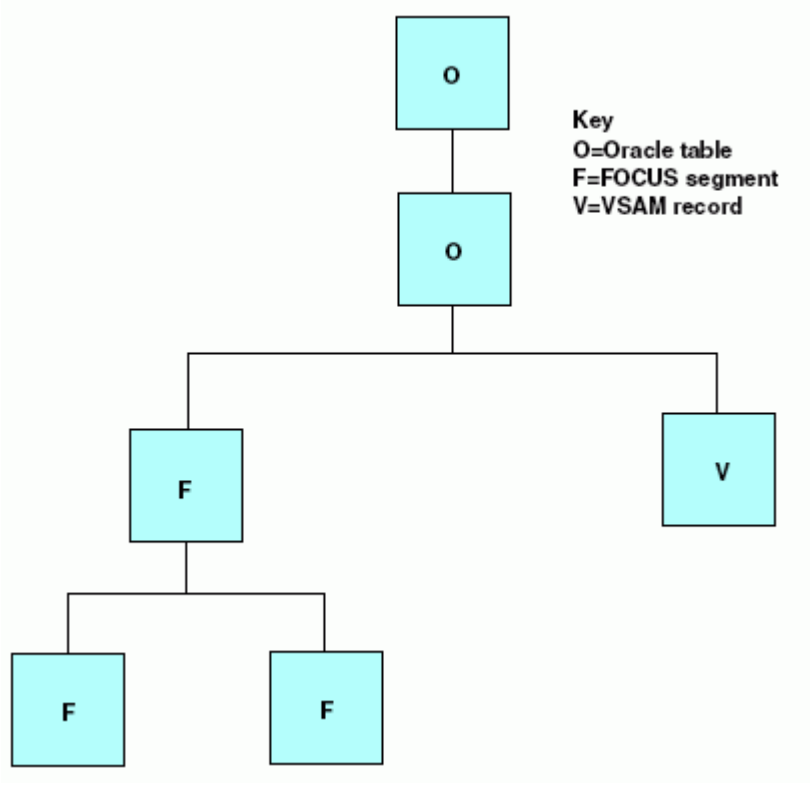
```
JOIN SUBPART IN AIRCRAFT TO PART IN AIRCRAFT AS SUB_PART
```

This produces the following data structure:



Relating Segments From Different Types of Data Sources

The JOIN command enables you to join segments from different data sources, creating temporary data structures containing related information from otherwise incompatible sources. For example, you can join two Oracle data sources to a FOCUS data source to a VSAM data source, as illustrated in the following diagram.



Joins between VSAM and fixed-format data sources are also supported in a Master File, as described in Chapter 7, *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 Data Source: An Alternate View

In this section:

Other Uses of an Alternate View

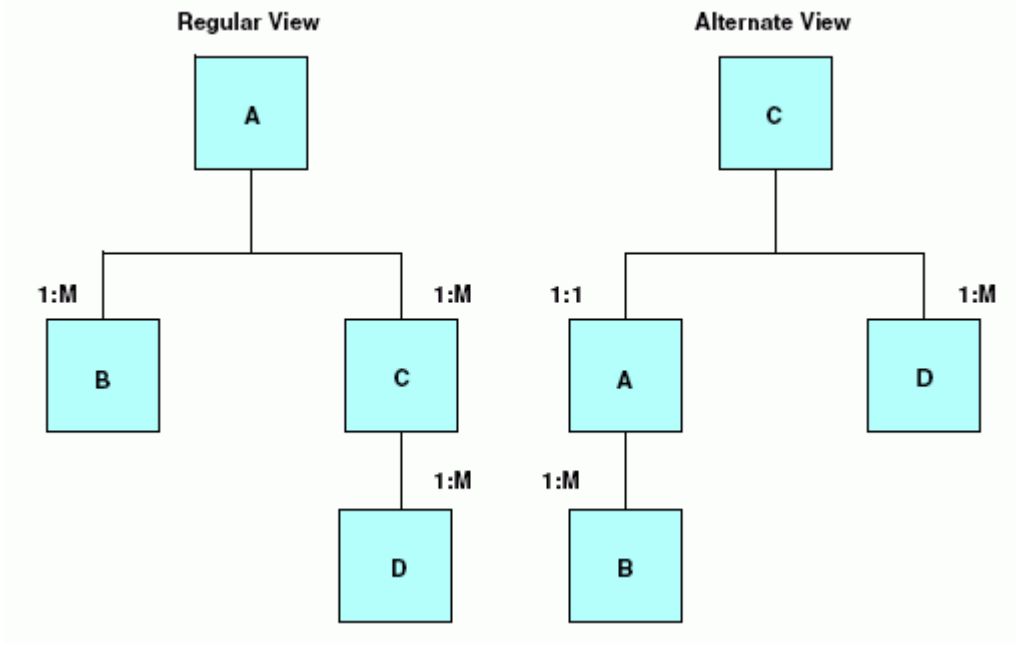
How to:

Specify an Alternate View

Example:

Specifying an Alternative View

If you are using a network data source or certain hierarchical data sources such as FOCUS, you can rotate the data source after you have described it. This creates an alternate view that changes some of the segment relationships and enables you to access the segments in a different order. Customizing the access path in this way makes it easier for a given application to access. This type of alternate view is illustrated in the following diagram.



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

Using an alternate view can be 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 can report from an alternate view that makes this the root segment; FOCUS then begins its record selection based on the relevant segment, and avoids reading irrelevant ancestral segments.

When you report from a data source using an alternate view, you can access the data more efficiently if both of the following conditions are satisfied:

- The field on which the alternate view is based is indexed. For FOCUS data sources, the alternate view field must include INDEX = I in the Master File.
- You use the field in a record selection test, with the WHERE or IF phrases, and make the selection criteria an equality or range test.

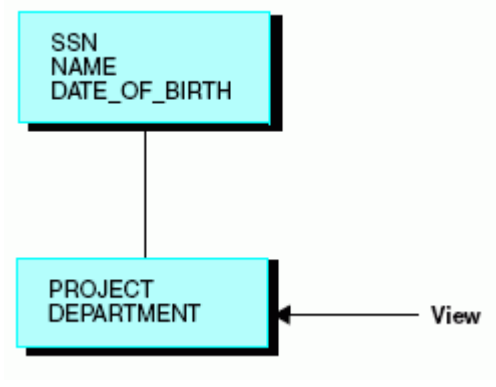
You can request an alternate view on any segment in a data source, except a cross-referenced segment. Request an alternate view with the TABLE command by naming a field from the segment you want to view as the new root segment. The only restriction on requesting an alternate view is that the field on which it is requested must be a real field in the data source. It cannot be a virtual field. This type of alternate view is illustrated in the following diagram.

Other Uses of an Alternate View

Other appropriate situations for using the alternate view capability with the file designer include:

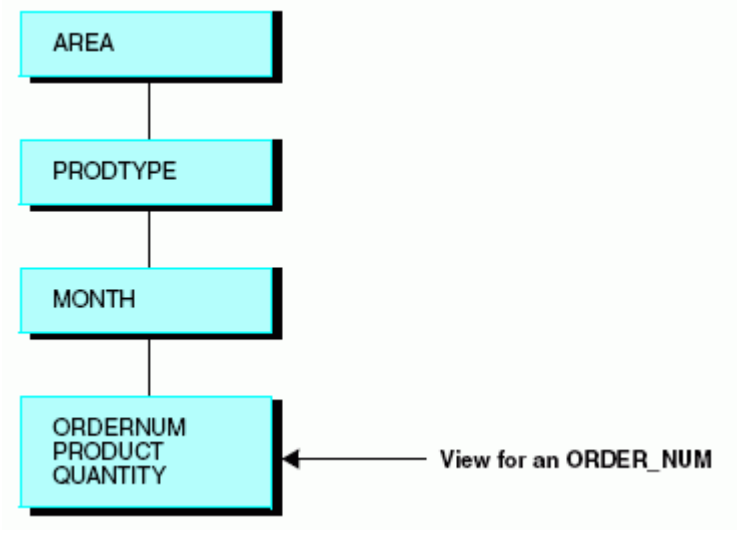
- Data sources with active individual record maintenance. The data sources can be structured for efficient update and management. For example, primary record keys can be placed at the top of the data source, even though requests frequently screen on other fields. The hierarchy does not assist in record 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 employee's current project is in a descendant segment. Access to all employees in a given department is obtained using the view from the department.



The following diagram further illustrates this type of alternate view.

- Individual records identified in descendant segments. You can use an alternate view to access a detail segment that is deep in the hierarchy. Consider a sales analysis situation. The data source 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 not often available, 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 data source from the vantage point of these 2,000 when the given child is involved in the screening in the request.

Note that retrieval of a given SSN can also be faster using a view, if the SSN values are indexed.

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

Syntax: How to Specify an Alternate View

Append a field name to the file name in the reporting command, using the syntax

```
TABLE FILE filename.fieldname
```

where:

filename

Is the name of the data source on which you are defining the alternate view.

fieldname

Is a field located in the segment that you are defining as the alternate root. It 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 test (such as EQ) or range test.

Example: Specifying an Alternative View

To report from the EMPLOYEE data source using an alternate view that makes the DEDUCT segment an alternate root, issue the following TABLE FILE command:

```
TABLE FILE EMPLOYEE.DED_CODE
```

FOCUS

4 Describing an Individual Field

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

Topics:

- ❑ Field Characteristics
- ❑ The Field's Name: FIELDNAME
- ❑ The Field Synonym: ALIAS
- ❑ The Displayed Data Type: USAGE
- ❑ The Stored Data Type: ACTUAL
- ❑ Null or MISSING Values: MISSING
- ❑ Describing an FML Hierarchy
- ❑ Validating Data: ACCEPT
- ❑ Online Help Information: HELPMESSAGE
- ❑ Alternative Report Column Titles: TITLE
- ❑ Documenting the Field: DESCRIPTION
- ❑ Multilingual Metadata
- ❑ Describing a Virtual Field: DEFINE
- ❑ Describing a Filter: FILTER
- ❑ Describing a Calculated Value: COMPUTE

Field Characteristics

The Master File describes the following field characteristics:

- The name of the field as identified in 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 how to interpret the field—that you can use as an alternative name in requests. This alternative name is defined by the ALIAS attribute.
- How the field stores and displays data, specified by the ACTUAL, USAGE, 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 data sources do not use the ACTUAL attribute, and instead use the USAGE attribute to describe the data as it is formatted.

  The USAGE attribute, also known by its alias, FORMAT, describes how a field is formatted when it appears in reports. You can also specify edit options such as date formats, floating dollar signs, and zero suppression. FOCUS handles the storage.

  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 data sources and most relational data sources.
- The option for a field to be virtual—rather than being stored in the data source—and have its value derived from information already in the data source. Virtual 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. See the *Developing Applications* manual.

The Field's Name: FIELDNAME

In this section:

Using a Long and Qualified Field Name

Using a Duplicate Field Name

Rules for Evaluating a Qualified Field Name

How to:

Identify the Field Name

Reference:

Usage Notes for FIELDNAME

Identify a field 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 data sources, you can assign any name to a field in a new data source.

When you generate a report, each column title in the report has the name of the field displayed in that column as its default, so assigning meaningful field names helps readers of the report. Alternatively, you can specify a different column title within a given report by using the AS phrase in the 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 158.

Syntax: How to Identify the Field Name

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

where:

field_name

Is the name you are giving this field. It can be a maximum of 66 characters. Some restrictions apply to names longer than 12 characters, as described below. The name can include any combination of letters, digits, and underscores (_), and must contain at least one 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 these can be used to refer to report columns, HOLD file fields, and other special objects.

If you must 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 158.

Reference: Usage Notes for FIELDNAME

Note the following rules when using FIELDNAME:

- **Alias.** FIELDNAME has an alias of FIELD.
- **Changes.** In a FOCUS data source, if the INDEX attribute has been set to I—that is, if an index has been created for the field—you cannot change the field name without rebuilding the data source. You may change the name in all other situations.

Using a Long and Qualified Field Name

How to:

Specify a Qualified Field Name in a Request

Change the Qualifying Character

Example:

Qualifying a Field Name

Reference:

Restrictions for 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, a useful technique when duplicate field names exist across segments in a Master File or in joined data sources. 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.

Temporary field names may also contain up to 66 characters. The names of text fields and indexed fields in FOCUS Master Files are limited to 12 characters. Text fields and indexed fields in XFOCUS Master Files are not subject to this 12 character limitation. However, the aliases for text and indexed fields may be up to 66 characters. Field names up to 66 characters appear as column titles in TABLE reports if there is no TITLE attribute or AS phrase.

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.

Syntax: How to Specify a Qualified Field Name in a Request

```
[filename.][segname.]fieldname
```

where:

filename

Is the name of the Master File or tag name. Tag names are used with the JOIN and COMBINE commands.

segname

Is the name of the segment in which the field resides.

fieldname

Is the name of the field.

Example: Qualifying a Field Name

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

```
EMPLOYEE.EMPINFO.EMP_ID
```

Syntax: How to Change the Qualifying Character

```
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.

Reference: Restrictions for Long and Qualified Field Names

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

- You cannot use a long name to specify a join:
  - In a JOIN command, for a cross-referenced field in a FOCUS data source.
  - In a multi-table Master File for a relational data source, for the KEYFLD and IXFLD attributes in the Access File.
- Indexed fields and text fields in FOCUS data sources cannot have field names longer than 12 characters. Indexed fields and text fields in XFOCUS data sources are not subject to this 12 character limitation. Long ALIAS names are supported for both types of data sources.
- The SQL Translator supports field names of up to 48 characters.
- A field name specified in an alternate file view cannot be long or qualified.
- 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 a Duplicate Field Name

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. The second occurrence is never accessed, and the following 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 the one retrieved.

Note: If a Master File includes duplicate field names for real fields and/or virtual fields, the following logic is used when retrieving a field:

- ❏ If only virtual fields are duplicated, the last virtual 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 virtual fields with the same name, the last virtual field is retrieved.
- ❏ If a field defined outside of a Master File has the same name as a virtual 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, they display qualified column titles for duplicate field names even when the request itself does not specify qualified names. The default value, OFF, disables qualified column titles.

Rules for Evaluating a Qualified Field Name

The following rules are used to evaluate qualified field names:

- The maximum field name qualification is *filename.segname.fieldname*. 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. 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. 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.

- 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. The value of the first field is retrieved, 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 is between two levels of qualification. To retrieve the CAR1 field, you must specify its alias.

The Field Synonym: ALIAS

In this section:

Implementing a Field Synonym

Example:

Using a Field Synonym

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 you choose, or, in special cases, a predefined 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 data source. After it has been assigned, you can use this alias in requests as a synonym for the regular field name. Assign this alternative name using the ALIAS attribute.

Example: Using a Field Synonym

In the EMPLOYEE data source, 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, USAGE = D12.2M,  $
```

Both names are equally valid within a request. The following TABLE requests illustrate that 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
```

Note: In extract files (HOLD, PCHOLD), the field name is used to identify fields, not the ALIAS.

Implementing a Field Synonym

The value you assign to ALIAS must conform to the same naming conventions to which the FIELDNAME attribute is subject, unless stated otherwise. 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 sources.** ALIAS describes a synonym, or alternative name, that you can use in a 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. The ALIAS attribute is required in the Master File, but it can have the value blank.

  Note that ALIAS is used in a different way for sequenced repeating fields, where its value is ORDER, as well as for RECTYPE and MAPVALUE fields when the data source includes multiple record types. For more information, see Chapter 5, *Describing a Sequential, VSAM, or ISAM Data Source*.
- **FOCUS data sources.** ALIAS describes a synonym, or alternative name, that you can use in a 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. The ALIAS attribute is required in the Master File, but it can have the value blank. For more information, see Chapter 6, *Describing a FOCUS Data Source*. Aliases can be changed without rebuilding the data source. If an alias is referred to in other data sources, similar changes may be needed in those Master Files.

The Displayed Data Type: USAGE

This attribute, which is also known as FORMAT, describes how to format a field when displaying it in a report or using it in a calculation.

For FOCUS data sources, which do not use the ACTUAL attribute, USAGE also specifies how to store the field. For other types of data sources, assign a USAGE value that corresponds to the ACTUAL value, to identify the field as the same data type used to store it in the data source. If the data is store as alphanumeric, assign the USAGE based on how the field will be displayed in your reports. The conversion is done automatically. For instructions on which ACTUAL values correspond to which USAGE values, see the documentation for the specific data adapter. For sequential, VSAM, and ISAM data sources, see Chapter 5, *Describing a Sequential, VSAM, or ISAM Data Source*.

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. Use these options to customize how the field appears in reports.

Syntax: How to Specify a Display Format

```
USAGE = tl[d]
```

where:

t

Is the data type. Valid values are A (alphanumeric), F (floating-point single-precision), D (floating-point double-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. The specification varies according to the data type. 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 USAGE value cannot exceed eight characters.

The values that you specify for type and field length determine the number of print positions allocated 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.

Note: If a numeric field cannot displya with the USAGE format given (for example, the result of aggregation is too large), asterisks appear.

See the sections for each format type for examples and additional information.

Reference: Usage Notes for USAGE

Note the following rules when using USAGE:

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

  For FOCUS data sources, 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. See *Numeric Display Options* on page 110 for additional information.
- **Alphanumeric.** 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.
- **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 validate dates automatically in transactions. Note that for some applications, such as assigning a date value using the DECODE function, you may wish instead to use alphanumeric, integer, or packed-decimal fields with date display options, which provide partial date functionality.
- **Date-Time.** The date-time format supports both the date and the time, similar to the timestamp data types available in many relational data sources. Date-time fields are stored in eight or ten bytes: four digits for date and either four or six digits for time, depending on whether the format specifies a microsecond. Computations only allow direct assignment within data types. All other operations are accomplished through a set of date-time functions.
- **Text.** Text fields can be used to store large amounts of data and display it with line breaks.

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 130.

The integer USAGE type is I. See *Numeric Display Options* on page 110. The format of the length specification is:

```
n
```

where:

n

Is the number of digits to display including a maximum of 10 digits and a leading minus sign if the field contains a negative value. The maximum integer value is 2147483647.

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 value composed of the digits zero to nine and an optional decimal point.

The floating-point double-precision USAGE type is D. See *Numeric Display Options* on page 110 for the compatible display options. The length specification format is:

```
t[.s]
```

where:

t

Is the number of characters to display including a maximum of 15 digits, an optional decimal point, and a leading minus sign if the field contains a negative value.

s

Is the number of digits that follow the decimal point. It can be up to 15 digits.

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 USAGE type is F. Compatible display options are described in *Numeric Display Options* on page 110. The length specification format is:

```
t[.s]
```

where:

t

Is the number of characters to display including a maximum of 7 digits, an optional decimal point, and a leading minus sign if the field contains a negative value.

s

Is the number of digits that follow the decimal point. It can be up to 7 digits.

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. See *Alphanumeric and Numeric Formats with Date Display Options* on page 130.

The packed-decimal USAGE type is P. The compatible display options are described in *Numeric Display Options* on page 110.

The length specification format is:

`m.n`

where:

m

Is the number of characters to display including a maximum of 31 digits, an optional decimal point, and a leading minus sign if the field contains a negative value.

n

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

For example:

Format	Display
P9.3	4168.368
P7	617542

Numeric Display Options

Example:

Using Numeric Display Options

Display options may be used to edit numeric formats. These options only affect how the data in the field is printed or appears on the screen, not how it is stored in your data source.

Edit Option	Meaning	Effect
-	Minus sign	Displays a minus sign to the right of negative numeric data. **Note:** Not supported with format options B, E, R, T, DMY, MDY, and YMD.
%	Percent sign	Displays a percent sign along with numeric data. Does not calculate the percent.
B	Bracket negative	Encloses negative numbers in parentheses.
C	Comma suppress	Suppresses the display of commas. Used with numeric format options M and N (floating and non-floating dollar sign) and data format D (floating-point double-precision).
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.

Edit Option	Meaning	Effect
M	Floating currency symbol ($ for US code page)	Places a floating currency symbol to the left of the highest significant digit. The default currency symbol depends on the code page. You can use the SET CURRSYMB=*symbol* command to specify any character as the currency symbol or one of the following currency codes: USD or '$' specifies U. S. dollars. GBP specifies the British pound. JPY specifies the Japanese yen. EUR specifies the Euro.
MDY	Month-Day-Year	Displays alphanumeric or integer data as a date in the form month/day/year.
N	Fixed currency symbol ($ for US code page)	Places a currency symbol to the left of the field. The symbol appears only on the first detail line of each page. The default currency symbol depends on the code page. You can use the SET CURRSYMB=*symbol* command to specify any character as the currency symbol or one of the following currency codes: USD or '$' specifies U. S. dollars. GBP specifies the British pound. JPY specifies the Japanese yen. EUR specifies the Euro.
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.

Example: Using Numeric Display Options

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

Option	Format	Data	Display
Minus sign	I2- D7- F7.2-	-21 -6148 -8878	21- 6148- 8878.00-
Percent sign	I2% D7% F3.2%	21 6148 48	21% 6,148% 48.00%
Comma suppression	D6c D7Mc D7Nc	41376 6148 6148	41376 $6148 $ 6148
Comma inclusion	I6C	41376	41,376
Zero suppression	D6S	0	
Bracket negative	I6B	-64187	(64187)
Credit negative	I8R	-3167	3167 CR
Leading zeroes	F4L	31	0031
Floating dollar	D7M	6148	$6,148
Non-floating dollar	D7N	5432	$ 5,432
Scientific notation	D12.5E	1234.5	0.12345D+04
Year/month/day	I6YMD I8YYMD	980421 19980421	98/04/21 1998/04/21
Month/day/year	I6MDY I8MDYY	042198 04211998	04/21/98 04/21/1998
Day/month/year	I6DMY I8DMYY	210498 21041998	21/04/98 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) 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 130.

Extended Currency Symbol Display Options

Reference:

Extended Currency Symbol Formats

You can select a currency symbol for display in report output regardless of the default currency symbol configured for National Language Support (NLS). Use the extended currency symbol format in place of the floating dollar (M) or non-floating dollar (N) display option. When you use the floating dollar (M) or non-floating dollar (N) display option, the currency symbol associated with the default code page is displayed. For example, when you use an American English code page, the dollar sign is displayed.

The extended currency symbol format allows you to display a symbol other than the dollar sign. For example, you can display the symbol for a United States dollar, a British pound, a Japanese yen, or the euro. Extended currency symbol support is available for numeric formats (I, D, F, and P). I

The extended currency symbol formats are specified as two-character combinations *in the last postitions* of any numeric display format. The first character in the combination can be either an exclamation point (!) or a colon (:). The colon is invariant across code pages, while the exclamation point is not:

Display Option	Description	Example
!d or :d	Fixed dollar sign.	D12.2:d
!D or :D	Floating dollar sign.	D12.2:D
!e or :e	Fixed euro symbol.	F9.2:e
!E or :E	Floating euro symbol on the left side.	F9.2:E
!F or :F	Floating euro symbol on the right side.	F9.2:F
!l or :l	Fixed British pound sign.	D12.1:l
!L or :L	Floating British pound sign.	D12.1:L
!y or :y	Fixed Japanese yen symbol.	I9:y
!Y or :Y	Floating Japanese yen symbol.	I9:Y

Reference: Extended Currency Symbol Formats

The following guidelines apply:

- ❏ A format specification cannot be longer than eight characters.
- ❏ The extended currency option must be the last option in the format.
- ❏ The extended currency symbol format cannot include the floating (M) or non-floating (N) display option.
- ❏ A non-floating currency symbol is displayed only on the first row of a report page. If you use field-based reformatting (as in the example that follows) to display multiple currency symbols in a report column, only the symbol associated with the first row is displayed. In this case, do not use non-floating currency symbols.
- ❏ Lowercase letters are transmitted as uppercase letters by the terminal I/O procedures. Therefore, the fixed extended currency symbols can only be specified in a procedure.
- ❏ Extended currency formats can be used with fields in floating point, decimal, packed, and integer formats. Alphanumeric and variable character formats cannot be used.

Example: Displaying Extended Currency Symbols

The following request displays the British pound sterling symbol on the row that represents England, the euro symbol on the row that represents Italy, and the Japanese yen symbol on the row that represents Japan.

```
SET PAGE-NUM = OFF
PRINT PRODNAME QUANTITY PRICE/D10.2!E
BY ORDER_DATE
WHERE QUANTITY GT 700;
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
TYPE = REPORT, GRID = OFF,$
ENDSTYLE
END
```

The output is:

Date Of Order:	Product Name:	Quantity:	Price:
2001/10/16	R5 Micro Digital Tape Recorder	726	€89.00
	ZT Digital PDA - Commercial	726	€499.00
2002/03/20	R5 Micro Digital Tape Recorder	751	€89.00
	ZT Digital PDA - Commercial	751	€499.00
2002/04/03	ZC Digital PDA - Standard	751	€299.00
2002/06/07	R5 Micro Digital Tape Recorder	751	€89.00
	ZT Digital PDA - Commercial	751	€499.00
	ZC Digital PDA - Standard	751	€299.00
	R5 Micro Digital Tape Recorder	702	€89.00
	ZT Digital PDA - Commercial	702	€499.00
	ZC Digital PDA - Standard	702	€299.00
2002/06/18	R5 Micro Digital Tape Recorder	751	€89.00
	ZT Digital PDA - Commercial	751	€499.00
2002/10/16	R5 Micro Digital Tape Recorder	798	€89.00
	ZT Digital PDA - Commercial	798	€499.00
2002/12/19	2 Hd VCR LCD Menu	701	€179.00

Alphanumeric Format

Reference:

Usage Notes for 4K Alphanumeric Fields

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 130.

The alphanumeric USAGE type is A. The format of the length specification is *n*, where *n* is the maximum number of characters in the field. You can have up to 3968 bytes in an alphanumeric field in a FOCUS file segment, and up to 4096 bytes in an XFOCUS file segment. You can have up to 4095 bytes in a fixed-format sequential data source. You may define the length in the Master File, a DEFINE FILE command, or a COMPUTE command.

For example:

Format	Display
A522	The minutes of today's meeting were submitted...
A2	B3
A24	127-A429-BYQ-49

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 you are displaying a product code in parts, with each part separated by a "-", include the following in a DEFINE command:

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

where:

fieldname

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

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

Reference: Usage Notes for 4K Alphanumeric Fields

- Long alphanumeric fields cannot be indexed.
- For FOCUS data sources, a segment still has to fit on a 4K page. Thus, the maximum length of an alphanumeric field depends on the length of the other fields within its segment.
- Long alphanumeric fields cannot be used in a CRTFORM.
- You can print or hold long alphanumeric fields, but you cannot view them online.
- Long alphanumeric fields may be used as keys.
- Long alphanumeric fields are not supported in Hot Screen.

Date Formats

Date format enables you to define a field as a date, then 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 appears.
- Perform 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 Display Options

Reference:

How Field Formats Y, YY, M, and W Are Stored

Date Literals Interpretation Table

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 USAGE specification. Otherwise it prints day of week.
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).

Display Option	Meaning	Effect
W	Day-of-Week	If it is included in a USAGE 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 USAGE specification, it prints the number of the day of the week (1-7; Mon=1).
w	Day-of-Week	Functions the same as uppercase W (described above), except that the first letter is uppercase and the following letters are lowercase.*
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.*
J[UL]	Julian format	Prints date in Julian format.
YYJ[UL]	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.

The following combinations of date components are not supported in date formats:

I2D, A2D, I2M, A2M, I2MD, A2MD

Reference: How Field Formats Y, YY, M, and W Are 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 do not sort based on DEFCENT and YRTHRESH settings. A field with a format of Y does 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.

Reference: Date Literals Interpretation Table

This table illustrates the behavior of 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	00/00d	00/0dd	00/ddd	0y/ddd
YYJUL	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	yy/ddd	*	*	*
YYJUL	CCyy/ddd	0yyy/ddd	yyyy/ddd	*
YY	*	*	*	*
Y	*	*	*	*
D	*	*	*	*
W	*	*	*	*

Note:

- CC stands for two century digits provided by DFC/YRT settings.
- * stands for message FOC177 (invalid date constant).
- Date literals are read from right to left. Date literals and fields can be used in computational expressions, as described in the *Creating Reports* manual.

Controlling the Date Separator

You can control the date separators when the date appears. In basic date format, such as YMD and MDYY, the date components appear separated by a slash character (/). The same is true for the year-month format, which appears 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 also be a period, a dash, or a blank, or can even be eliminated entirely. The following table shows the USAGE specifications for changing 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.)

Note:

- ❏ You can change the date separator in the following date formats: YYMD, MDYY, DMYY, YMD, MDY, DMY, YYM, MYY, YM, MY, YYQ, QYY, YQ, and QY.
- ❏ You cannot change the date separator in a format that includes date translation options.
- ❏ You cannot change the date separator (/) in an alphanumeric or numeric format with date display options (for example, I8YYMD).

Date Translation

Example:

Using a Date Format

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

Example: Using a Date Format

The following chart shows 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 appears.

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 non-FOCUS data source. If the ACTUAL format does not specify the order of the month, day, and year, it is inferred from the USAGE format.

Using a Date Field

A field formatted as a date is automatically validated when 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 enable you to specify a date in a natural, easily understandable way, by including spaces between date components and using abbreviations of month names. 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 all methods of data source updating. The following chart shows examples:

```
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 the 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 USAGE specification of the target field. Date components are separated by one or more blanks.

For example, if a USAGE 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 are both 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 USAGE specification. In addition, the numeric date literal must include all of the date components included in the USAGE specification. For example, if the USAGE 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 data source 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. Y or YY, Q, M, W, and D are valid components.

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 USAGE 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 a Date Field

How to:

Convert a Date Field

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 USAGE 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 99, ORDERDATE is 99.

You can also assign the value of ORDERDATE to REPORTDATE; if the value of ORDERDATE is 99, the value of REPORTDATE is Jan 1 99. In this case, REPORTDATE is given values for the missing date components.

Syntax: How to Convert a Date Field

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

where:

field1

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

format

Is the USAGE (or FORMAT) 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 (alphanumeric, integer, or date) and the date components (YY, Y, Q, M, W, D) of *field1* and *field2* do not need to match.

How a Date Field Is Represented Internally

Date fields are represented internally as four-byte binary integers indicating the time elapsed 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 USAGE 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 enter the numeric literal for February 13, 1964 (that is, 021364), and then print the field in a report, 02/13/64 appears. If you use it in the equation:

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

then the value of DAYS is 15. However, the internal representation of REPORTDATE is 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.

To display blanks or the actual base date in a report, 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 appear as blanks, and all date fields that are entered blank or as all zeroes are accepted during validation and interpreted as the base date. They appear as blanks, but are interpreted as the base date in date computations and expressions.

Displaying a Non-Standard Date Format

How to:

Invoke ALLOWCVTERR

By default, if a date field in a non-FOCUS data source contains an invalid date, a message appears and the entire record fails to appear in a report. For example, if a date field contains '980450' with an ACTUAL of A6 and a USAGE of YMD, the record containing that field does not appear. The SET ALLOWCVTERR command enables you to display the rest of the record that contains the incorrect date.

Syntax: How to Invoke ALLOWCVTERR

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

where:

ON
: Enables you to display a field containing an incorrect date.

OFF
: Generates a diagnostic message if incorrect data is encountered, and does not display the record containing the bad data. OFF is the default value.

When a bad date is encountered, ALLOWCVTERR sets the value of the field to either MISSING or to the base date, depending on whether 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 affects only how the base date appears. See the *Developing Applications* manual for a description of DATEDISPLAY.

Date Format Support

Date format fields are used in special ways with the following 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 difference between the field's face value and the standard base date. Date fields in SAVE files and formatted HOLD files (for example, USAGE WP) are stored without any display options.

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

- **FML.** 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 USAGE can be:

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

The number of a month (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 displays the value 4 as APR, for example. This is particularly useful in reports where columns or rows are sorted by month. They 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 is interpreted as an integer with a floating dollar sign.)

Date-Time Formats

How to:

Enable ISO Standard Date-Time Notation

Example:

Using SET DTSTANDARD

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

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

Computations only allow direct assignment within data types: alpha to alpha, numeric to numeric, date to date, and date-time to date-time. All other operations are accomplished through a set of date-time functions. See the *Using Functions* manual for information on subroutines for manipulating date-time fields.

Date-time formats can also produce output values and accept input values that are compatible with the ISO 8601:2000 date-time notation standard. A SET parameter and specific formatting options enable this notation.

Syntax: How to Enable ISO Standard Date-Time Notation

```
SET DTSTANDARD = {OFF|ON|STANDARD|STANDARDU}
```

where:

`OFF`

Does not provide compatibility with the ISO 8601:2000 date-time notation standard. OFF is the default value.

`ON|STANDARD`

Enables recognition and output of the ISO standard formats, including use of T as the delimiter between date and time, use of period or comma as the delimiter of fractional seconds, use of Z at the end of "universal" times, and acceptance of inputs with time zone information. STANDARD is a synonym for ON.

`STANDARDU`

Enables ISO standard formats (like STANDARD) and also, where possible, converts input strings to the equivalent "universal" time (formerly known as "Greenwich Mean Time"), thus enabling applications to store all date-time values in a consistent way.

Example: Using SET DTSTANDARD

The following request displays date-time values input in ISO 8601:2000 date-time standard formats. With SET DTSTANDARD=OFF, the request terminates with a (FOC177): INVALID DATE CONSTANT:

```
SET DTSTANDARD = &STAND
DEFINE FILE EMPLOYEE
-* The following input is format YYYY-MM-DDThh:mm:ss.sTZD
DT1/HYYMDs  =     DT(2004-06-01T19:20:30.45+01:00);
-* The following input has comma as the decimal separator
DT2/HYYMDs  =     DT(2004-06-01T19:20:30,45+01:00);
DT3/HYYMDs  =     DT(20040601T19:20:30,45);
DT4/HYYMDUs =     DT(2004-06-01T19:20:30,45+01:00);
END
TABLE FILE EMPLOYEE
HEADING CENTER
"DTSANDARD = &STAND "
" "
SUM CURR_SAL NOPRINT DT1 AS 'DT1: INPUT = 2004-06-01T19:20:30.45+01:00'
OVER DT2 AS 'DT2: INPUT = 2004-06-01T19:20:30,45+01:00'
OVER DT3 AS 'DT3: INPUT = 20040601T19:20:30,45'
OVER DT4 AS 'DT4: OUTPUT  FORMAT HYYMDUs'
END
```

With DTSTANDARD= STANDARD, The shows that the input values were accepted, but the time zone offsets in DT1, DT2, and DT4 (+01:00) were ignored on output. The character U in the format for DT4 causes the T separator to be used between the date and the time:

```
                        DTSANDARD = STANDARD

DT1: INPUT = 2004-06-01T19:20:30.45+01:00  2004-06-01 19:20:30.450
DT2: INPUT = 2004-06-01T19:20:30,45+01:00  2004-06-01 19:20:30.450
DT3: INPUT = 20040601T19:20:30,45          2004-06-01 19:20:30.450
DT4: OUTPUT  FORMAT HYYMDUs                2004-06-01T19:20:30.450
```

With DTSTANDARD= STANDARDU, The output shows that the values DT1, DT2, and DT4 were converted to universal time by subtracting the time zone offsets (+01:00):

```
                        DTSANDARD = STANDARDU

DT1: INPUT = 2004-06-01T19:20:30.45+01:00  2004-06-01 18:20:30.450
DT2: INPUT = 2004-06-01T19:20:30,45+01:00  2004-06-01 18:20:30.450
DT3: INPUT = 20040601T19:20:30,45          2004-06-01 19:20:30.450
DT4: OUTPUT  FORMAT HYYMDUs                2004-06-01T18:20:30.450
```

Describing a Date-Time Field

How to:

Describe a Numeric Date-Time Value Without Display Options

Describe a Time-Only Value

Describe a Date-Time Value

Reference:

Display Options for a Time-Only Value

Display Options for the Date Component of a Date-Time Field

Display Options for the Time Component of a Date-Time Field

Date-Time Usage Notes

In a Master File, the USAGE (or FORMAT) attribute determines how date-time field values appear in report output and forms, and how they behave in expressions and functions. For FOCUS data sources, it also determines how they are stored.

Format type H describes date-time fields. The USAGE attribute for a date-time field contains the H format code and can identify either the length of the field or the relevant date-time display options.

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

Syntax: How to Describe a Numeric Date-Time Value Without Display Options

This format is appropriate for alphanumeric HOLD files or transaction files.

```
USAGE = Hn
```

where:

n

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

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

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

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

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

Syntax: How to Describe a Time-Only Value

```
USAGE = Htimefmt1
```

where:

timefmt1

Is the USAGE format for displaying time only. Hour, minute, and second components are always separated by colons (:), with no intervening blanks. For information, see *Display Options for a Time-Only Value* on page 135.

Reference: Display Options for a Time-Only Value

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

Option	Meaning	Effect
`H`	Hour (two digits). If the format includes the option a or A, the hour value is from 01 to 12. Otherwise, the hour value is from 00 to 23, with 00 representing midnight.	Prints a two-digit hour. For example: `USAGE = HH` prints `02`
`h`	Hour with zero suppression. If the format includes the option a or A, the hour value is from 1 to 12. Otherwise, the hour is from 0 to 23.	Displays the hour with zero suppression. For example: `USAGE = Hh` prints `2`
`I`	Minute (two digits). The minute value is from 00 to 59.	Prints the two-digit minute. For example: `USAGE = HHI` prints `02:05`
`i`	Minute with zero suppression. The minute value is from 0 to 59.	Prints the minute with zero suppression. This cannot be used together with an hour format (H or h). For example: `USAGE = Hi` prints `5`
`S`	Second (two digits). 00 to 59	Prints the two-digit second. For example: `USAGE = HHIS` prints `02:05:27`
`s`	Millisecond (three digits, after the decimal point in the second). 000 to 999	Prints the second to three decimal places. For example: `USAGE = HHISs` prints `02:05:27.123`

Option	Meaning	Effect
m	Microsecond (three additional digits after the millisecond). 000 through 999	Prints the second to six decimal places. For example: `USAGE = HSsm` prints `27.123456`
A	12-hour time display with AM or PM in uppercase.	Prints the hour from 01 to 12, followed by AM or PM. For example: `USAGE = HHISA` prints `02:05:27AM`
a	12-hour time display with am or pm in lowercase.	Prints the hour from 01 to 12, followed by am or pm. For example: `USAGE = HHISa` prints `02:05:27am`
Z	24-hour time display with Z to indicate universal time. Z is incompatible with AM/PM output.	Prints the hour from 01 to 24, followed by Z. For example: `USAGE = HHISZ` prints `14:30[:20.99]Z`

When the format includes more than one time display option:

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

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

Syntax: How to Describe a Date-Time Value

```
USAGE = Hdatefmt [separator] [timefmt2]
```

where:

datefmt

Is the USAGE format for displaying the date portion of the date-time field. For information, see *Display Options for the Date Component of a Date-Time Field* on page 138.

separator

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

With the STANDARD and STANDARDU settings, the separator for dates is always hyphen. The separator between date and time is blank by default. However, if you specify the character U as the separator option, the date and time will be separated by the character T.

timefmt2

Is the format for a time that follows a date. Time is separated from the date by a blank; time components are separated from each other by colons. Unlike the format for time alone, a time format that follows a date format consists of at most two characters: a single character to represent all of the time components that appear and, optionally, one character for an AM/PM option. For information, see *Display Options for the Time Component of a Date-Time Field* on page 139.

Reference: Display Options for the Date Component of a Date-Time Field

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

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

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

Reference: Display Options for the Time Component of a Date-Time Field

The following table lists the valid options. Assume the date is February 5, 1999 and the time is 02:05:25.444555 a.m.

Option	Meaning	Example
`H`	Prints hour.	`USAGE = HYYMDH` prints `1999/02/05 02`
`I`	Prints hour:minute.	`USAGE = HYYMDI` prints `1999/02/05 02:05`
`S`	Prints hour:minute:second.	`USAGE = HYYMDS` prints `1999/02/05 02:05:25`
`s`	Prints hour:minute:second.millisecond.	`USAGE = HYYMDs` prints `1999/02/05 02:05:25.444`
`m`	Prints hour:minute:second.microsecond.	`USAGE = HYYMDm` prints `1999/02/05 02:05:25.444555`
`A`	Prints AM or PM. This uses the 12-hour system and causes the hour to be printed with zero suppression.	`USAGE = HYYMDSA` prints `1999/02/05 2:05:25AM`
`a`	Prints am or pm. This uses the 12-hour system and causes the hour to be printed with zero suppression.	`USAGE = HYYMDSa` prints `1999/02/05 2:05:25am`
`Z`	Prints Z to indicate universal time. This uses the 24-hour system. Z is incompatible with AM/PM output.	`USAGE = HHISZ` prints `14:30[:20.99]Z`

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

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

Reference: Date-Time Usage Notes

- ❏ In order to have a time component, you must have a day component.
- ❏ If you use the k option, you cannot change the date separator.

Character Format A*n*V

How to:

Specify AnV Fields in a Master File

Example:

Specifying the AnV Format in a Master File

Reference:

Usage Notes for AnV Format

Propagating an AnV Field to a HOLD File

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

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

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

Note: Because of the two bytes of overhead and the additional processing required to strip them, A*n*V format is not recommended for use in non-relational data sources.

Syntax: How to Specify A*n*V Fields in a Master File

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

where:

n

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

Note: HOLD FORMAT ALPHA creates an ACTUAL format of A*n*W in the Master File. See *Propagating an AnV Field to a HOLD File* on page 142.

Example: Specifying the A*n*V Format in a Master File

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

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

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

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

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

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

or

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

or

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

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

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

or

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

or

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

Reference: Usage Notes for A*n*V Format

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

Reference: Propagating an A*n*V Field to a HOLD File

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

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

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

For example, the A39V field named TITLEV, is propagated to the HOLD file as:

```
FIELDNAME = TITLEV ,E03 ,A39V ,A39W ,$
```

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

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

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

For example, the A39V field named TITLEV, is propagated to a HOLD FORMAT DB2 file as:

```
FIELDNAME = 'TITLEV', 'TITLEV', A39V, A39V ,$
```

Text Field Format

Reference:

Usage Notes for Text Field Format

```
FIELD = fieldname, ALIAS = aliasname, USAGE = TXn[F],$
```

where:

fieldname

Is the name you assign the text field.

aliasname

Is an alternate name for the field name.

n

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 appears. When F is not specified, the field reappears exactly as entered.

For example, the text field in the COURSES data source 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 data sources.
TX35	This course provides the DP professional with the skills needed to create, maintain, and report from FOCUS data sources.

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

Reference: Usage Notes for Text Field Format

- ❏ Conversion between text and alphanumeric fields is supported in DEFINE and COMPUTE commands.
- ❏ Multiple text fields are supported, and they and be anywhere in the segment. segment.

The Stored Data Type: ACTUAL

In this section:

ACTUAL Attribute

How to:

Specify the ACTUAL Attribute

Reference:

ACTUAL to USAGE Conversion

COBOL Picture to USAGE Format Conversion

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.

ACTUAL Attribute

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

Syntax: How to Specify the ACTUAL Attribute

```
ACTUAL = format
```

where:

format

Consists of values taken from the following table, which shows the codes for the types of data that can be 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* = 1-4095 for fixed-format sequential and VSAM data sources, and 1-256 for other non-FOCUS data sources. Alphanumeric characters A-Z, 0-9, and the special characters in the EBCDIC display mode. A*n* accepts all the date-time string formats, as well as the H*n* display formats. ACTUAL=A*n* also accepts a date-time field as it occurs in an alphanumeric HOLD file or SAVE file.
D8	Double-precision, floating-point numbers, stored internally in eight bytes.
F4	Single-precision, floating-point numbers, stored internally in four bytes.
H*n*	H8, H10, or H12 accepts a date-time field as it occurs in a binary HOLD file or SAVB file.
I*n*	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* = 1-16. 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.

ACTUAL Type	Meaning
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. If the field contains an assumed decimal point, represent the field with an ACTUAL format of Z*n* and a USAGE format of P*m*.*d*, where *m* is the total number of digits in the display plus the assumed decimal point, *d* is the number of decimal places, and *m* must be at least 1 greater than the value of *n*. For example, a field with ACTUAL=Z5 and one decimal place needs USAGE=P6.1 (or P7.1, or greater).

Note:

- Unless your data source is created by a program, all of the characters are either of type A (alphanumeric) or type Z (zoned decimal).
- ACTUAL formats supported for date-time values are A*n*, H8, H10, and H12. A*n* accepts all the date-time string formats as well as the H*n* USAGE display format. ACTUAL=H8, H10, or H12 accepts a date-time field as it occurs in a binary HOLD file or SAVB file. ACTUAL=A*n* accepts a date-time field as it occurs in an alphanumeric HOLD file or SAVE file.
- If you create a binary HOLD file from a data source with a date-time field, the ACTUAL format for that field is of the form H*n*. If you create an alphanumeric HOLD file from a data source with a date-time field, the ACTUAL format for that field is of the form A*n*.

Reference: ACTUAL to USAGE Conversion

The following conversions from ACTUAL format to USAGE (display) format are automatically handled and do not require invoking a function:

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

Reference: COBOL Picture to USAGE Format Conversion

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

COBOL USAGE FORMAT	BYTES OF COBOL PICTURE	INTERNAL STORAGE	ACTUAL FORMAT	USAGE FORMAT
DISPLAY DISPLAY DISPLAY DISPLAY	X(4) S99 9(5)V9 99	4 2 6 2	A4 Z2 Z6.1 A2	A4 P3 P8.1 A2
COMP COMP COMP* COMP COMP-1**	S9 S9(4) S9(5) S9(9) —	4 4 4 4 4	I2 I2 I4 I4 F4	I1 I4 I5 I9 F6
COMP-2***	—	8	D8	D15
COMP-3 COMP-3 COMP-3	9 S9V99 9(4)V9(3)	8 8 8	P1 P2 P4	P1 P5.2 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, VSAM, and ISAM data sources, see Chapter 5, *Describing a Sequential, VSAM, or ISAM Data Source*. For other types of data sources, see the documentation for the specific data adapter. Note that FOCUS data sources do not use the ACTUAL attribute, and instead rely upon the USAGE attribute to specify both how a field is stored and formatted.

Null or MISSING Values: MISSING

In this section:

Using a Missing Value

How to:

Specify a Missing Value

Reference:

Usage Notes for 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 data sources 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:

- **Create new segment instances.** If no value is supplied for a field for which MISSING has been turned ON in the Master File or in a DEFINE or COMPUTE definition, then the field is assigned a missing value.
- **Generate reports.** If a field with a null value is retrieved, the field value is not used in aggregating calculations such as averaging and summing. If the report calls for the field's value to display, a special character appears to indicate 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 or the SET HNODATA command for HOLD files, as described in the *Developing Applications* manual.

Syntax: How to Specify a Missing Value

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

where:

ON

Distinguishes a missing value from an intentionally entered blank or zero when creating new segment instances and reporting.

OFF

Does not distinguish between missing values and blank or zero values when creating new segment instances and reporting. OFF is the default value.

Reference: Usage Notes for MISSING

Note the following rules when using MISSING:

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

  FOCUS data sources also support MISSING=ON, which assigns numeric fields the value -9998998 for NULL values and alphanumeric fields the value '.'.
- **Changes.** You can change the MISSING attribute at any time. Note that changing MISSING does not affect the actual stored data values that were entered using the old setting. However, it does affect how that data is interpreted: if null data is entered when MISSING is turned ON, and then MISSING is switched to OFF, the data originally entered as null is interpreted as blanks (for alphanumeric fields) or zeroes (for numeric fields). The only exception is FOCUS data sources, in which the data originally entered as missing is interpreted as the internal missing value for that data type, which is described in Chapter 6, *Describing a FOCUS Data Source*.

Using a Missing Value

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, a value of zero is automatically be supplied for the two blank records. Thus, the average of these four records is (0+0+1+3)/4, or 1. If you turn MISSING to ON, the two blank records are not used in the calculation, so the average is (1+3)/2, or 2.

Missing values in a unique segment are also automatically supplied with a zero, a blank, or a missing value depending on the MISSING attribute. What distinguishes missing values in unique segments from other values 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. It includes alternative ways of distinguishing these values in reports, such as using the WHERE phrase with MISSING selection operators, and creating virtual fields using the DEFINE FILE command with the SOME or ALL phrase.

Describing an FML Hierarchy

How to:

Specify a Hierarchy Between Fields in a Master File

Assign Descriptive Captions for Hierarchy Field Values

Example:

Defining a Hierarchy in a Master File

The Financial Modeling Language (FML) supports dynamic reporting against hierarchical data structures.

You can define the hierarchical relationships between fields in a Master File and automatically display these fields using FML. You can also provide descriptive captions to appear in reports in place of the specified hierarchy field values.

In the Master File, use the PROPERTY=PARENT_OF and REFERENCE=*hierarchyfld* attributes to define the hierarchical relationship between two fields.

The parent and child fields must have the same FORMAT or USAGE, and their relationship should be hierarchical. The formats of the parent and child fields must both be numeric or both alphanumeric.

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

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

where:

parentfield

Is the parent field in the hierarchy.

PROPERTY=PARENT_OF

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

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

seg

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

hierarchyfld

Is the child field in the hierarchy.

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

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

Syntax: How to Assign Descriptive Captions for Hierarchy Field Values

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

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

where:

captionfield

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

PROPERTY=CAPTION

Signifies that this field contains a descriptive caption that appears in place of the hierarchy field values.

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

seg

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

hierarchyfld

Is the hierarchy field.

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

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

Example: Defining a Hierarchy in a Master File

The CENTGL Master File contains a chart of accounts hierarchy. The field GL_ACCOUNT_PARENT is the parent field in the hierarchy. The field GL_ACCOUNT is the hierarchy field. The field GL_ACCOUNT_CAPTION can be used as the descriptive caption for the hierarchy field:

```
FILE=CENTGL     ,SUFFIX=FOC
SEGNAME=ACCOUNTS,SEGTYPE=S01
FIELDNAME=GL_ACCOUNT,           ALIAS=GLACCT,  FORMAT=A7,
          TITLE='Ledger,Account', FIELDTYPE=I, $
FIELDNAME=GL_ACCOUNT_PARENT,    ALIAS=GLPAR,   FORMAT=A7,
          TITLE=Parent,
          PROPERTY=PARENT_OF, REFERENCE=GL_ACCOUNT, $
FIELDNAME=GL_ACCOUNT_TYPE,      ALIAS=GLTYPE,  FORMAT=A1,
          TITLE=Type,$
FIELDNAME=GL_ROLLUP_OP,         ALIAS=GLROLL,  FORMAT=A1,
          TITLE=Op, $
FIELDNAME=GL_ACCOUNT_LEVEL,     ALIAS=GLLEVEL, FORMAT=I3,
          TITLE=Lev, $
FIELDNAME=GL_ACCOUNT_CAPTION,   ALIAS=GLCAP,   FORMAT=A30,
          TITLE=Caption,
          PROPERTY=CAPTION,   REFERENCE=GL_ACCOUNT, $
FIELDNAME=SYS_ACCOUNT,          ALIAS=ALINE,   FORMAT=A6,
          TITLE='System,Account,Line', MISSING=ON, $
```

Validating Data: ACCEPT

How to:

Validate Data

Reference:

Usage Notes for 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 data sources may use the ACCEPT attribute to specify multiple RECTYPE values, which are discussed in Chapter 5, *Describing a Sequential, VSAM, or ISAM Data Source*.

Syntax: How to Validate Data

```
ACCEPT = list
ACCEPT = value1 TO value2
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.

value1 TO value2

Gives the range of acceptable values. 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 data sources. For more information, see Chapter 6, *Describing a FOCUS Data Source*.

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 are 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 rules when using ACCEPT:

- **Alias.** ACCEPT does not have an alias.
- **Changes.** You can change the information in an ACCEPT attribute at any time.
- **Virtual fields.** You cannot use the ACCEPT attribute to validate virtual fields created with the DEFINE attribute.
- **HOLD files.** If you wish to propagate the ACCEPT attribute 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

In this section:

Setting a HELP (PF) Key

How to:

Include Online Help Information in a Master File

HELPMESSAGE is an optional field attribute. It enables you to include a one-line text message in the Master File. This text, or message, appears 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 appears 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.

Syntax: How to Include Online Help Information in a Master File

The syntax for the HELPMESSAGE attribute in the Master File is:

```
FIELDNAME = name, ALIAS = alias, USAGE = 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, USAGE = 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 message appears. Then the specified HELPMESSAGE text appears:

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

Note:

- Messages appear whether or not the HELPMESSAGE attribute is used. The attribute also causes a message to appear 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, USAGE = YMD,
     HELPMESSAGE = THE FORMAT FOR HIRE_DATE IS YMD,$
  ```

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

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

- The same message provided with the HELPMESSAGE attribute appears 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 syntax, which is the alias for HELPMESSAGE:

```
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, the following message appears:

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

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

Alternative Report Column Titles: TITLE

How to:

Specify an Alternative Title

Reference:

Usage Notes for TITLE

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

You can also 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.

Master Files support TITLE attributes for multiple languages. For information, see *Multilingual Metadata* on page 161.

Syntax: How to Specify an Alternative Title

```
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 (,). 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 if it includes commas or leading blanks.

For example:

```
FIELD = LNAME, ALIAS = LN, USAGE = 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 turn it off with the SET TITLES=OFF command.
- **Virtual fields.** If you use the TITLE attribute for a virtual field created with the DEFINE attribute, the semicolon (;) terminating the DEFINE expression must be on the same line as the TITLE keyword.
- **HOLD files.** To propagate the TITLE attribute 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

How to:

Supply Field Documentation

Reference:

Usage Notes for 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 2K (2048) 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 Master File are described in Chapter 1, *Understanding a Data Source Description*.

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

Master Files support description attributes for multiple languages. For information, see *Multilingual Metadata* on page 161.

Syntax: How to Supply Field Documentation

```
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 2K (2048) characters. If it contains a comma, the string must be enclosed within single quotation marks.

For example:

```
FIELD=UNITS,ALIAS=QTY,USAGE=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.
- **Virtual fields.** You can use the DESCRIPTION attribute for a virtual field created with the DEFINE attribute.

Multilingual Metadata

How to:

Specify Multilingual Metadata in a Master File

Activate the Use of a Language

Reference:

Languages and Language Code Abbreviations

Usage Notes for Multilingual Metadata

Example:

Using Multilingual Titles in a Request

Using Multilingual Descriptions in a Master File

Master Files support column headings and descriptions in multiple languages. The heading or description used depends on the value of the LANG parameter and whether a TITLE_*ln* or DESC_*ln* attribute is specified in the Master File, where *ln* identifies the language to which the column heading or description applies.

In a Master File, column headings are taken from:

1. A heading specified in the report request using the AS phrase.
2. A TITLE attribute in the Master File, if no AS phrase is specified in the request and SET TITLES=ON.
3. The field name specified in the Master File, if no AS phrase or TITLE attribute is specified, or if SET TITLES=OFF.

Syntax: How to Specify Multilingual Metadata in a Master File

```
FIELDNAME = field, ...
    .
    .
    .
TITLE= default_column_heading
TITLE_ln = column_heading_for_ln
    .
    .
    .
    .
    .
DESC= default_desc
DESC_ln = desc_for_ln
    .
    .
    .
```

where:

field

Is a field in the Master File.

default_column_heading

Is the column heading to use when SET TITLES=ON and either the LANG parameter is set to the default language for the server, or another language is set but the Master File has no corresponding TITLE_*ln* attribute for that field. This column heading is also used if an the *ln* value is invalid.

default_desc

Is the description to use when either the LANG parameter is set to the default language for the server, or another language is set but the Master File has no corresponding DESC_*ln* attribute for that field. This description is also used if an the *ln* value is invalid.

TITLE_*ln* = *column_heading_for_ln*

Specifies the language for which the column heading applies and the text of the column heading in that language. That column heading is used when SET TITLES=ON, the LANG parameter is set to a non-default language for the server, and the Master File has a corresponding TITLE_*ln* attribute, where *ln* is the two-digit code for the language specified by the LANG parameter Valid values for *ln* are the two-letter ISO 639 language code abbreviations. For information, see *Languages and Language Code Abbreviations* on page 163

`DESC_ln = desc_for_ln`

Specifies the language for which the description applies and the description text in that language. This description is used when the LANG parameter is set to a non-default language for the server and the Master File has a corresponding DESC_*ln* attribute. Valid values for *ln* are the two-letter ISO 639 language code abbreviations.

Reference: Languages and Language Code Abbreviations

Language Name	Two-Letter Language Code	Three-Letter Language Abbreviation
Arabic	ar	ARB
Baltic	lt	BAL
Chinese - Simplified GB	zh	PRC
Chinese - Traditional Big-5	tw	ROC
Czech	cs	CZE
Danish	da	DAN
Dutch	nl	DUT
English - American	en	AME or ENG
English - UK	uk	UKE
Finnish	fi	FIN
French - Canadian	fc	FRE
French - Standard	fr	FRE
German - Austrian	at	GER
German - Standard	de	GER
Greek	el	GRE
Hebrew	iw	HEW
Italian	it	ITA
Japanese - Shift-JIS(cp942) on ascii cp939 on EBCDIC	ja	JPN

Language Name	Two-Letter Language Code	Three-Letter Language Abbreviation
Japanese - EUC(cp10942) on ascii (UNIX)	je	JPE
Korean	ko	KOR
Norwegian	no	NOR
Polish	pl	POL
Portuguese - Brazilian	br	POR
Portuguese - Portugal	pt	POR
Russian	ru	RUS
Spanish	es	SPA
Swedish	sv	SWE
Thai	th	THA
Turkish	tr	TUR

Syntax: How to Activate the Use of a Language

Issue the following command in a supported profile, on the command line, or in a FOCEXEC:

```
SET LANG = lng
```

or

```
SET LANG = ln
```

In the NLSCGF ERRORS configuration file, issue the following command

```
LANG = lng
```

where:

lng
Is the three-letter abbreviation for the language.

ln
Is the two-letter ISO language code.

Note: If SET LANG is used in a procedure, its value will override the values set in NLSCGF ERRORS or in any profile.

Reference: Usage Notes for Multilingual Metadata

- To generate the correct characters, all languages used must be on the code page specified at startup.
- Master Files should be stored using the code page used by FOCUS.
- Multilingual descriptions are supported with all fields described in the Master File, including DEFINE and COMPUTE fields.
- The user must create the NLSCFG file. On z/OS, the NLSCFG file must be a member in the concatenation of data sets allocated to DDNAME ERRORS. On z/VM, it must have filetype ERRORS.
- If you issue a HOLD command with SET HOLDATTR=ON, only one TITLE attribute is propagated to the HOLD Master File. Its value is the column heading that would have appeared on the report output.
- After it is referenced in a request, the Master File is stored in memory. If you want to produce a report against the same Master File but generate column titles in a different language, you must make sure the Master File is re-read after you issue the SET LANG command and prior to running the request. You can make sure the Master File is re-read by issuing a CHECK FILE command for that Master File.

Example: Using Multilingual Descriptions in a Master File

The following Master File for the CENTINV data source specifies French descriptions (DESC_FR) and Spanish descriptions (DESC_ES) as well as default descriptions (DESC) for the PROD_NUM and PRODNAME fields:

```
FILE=CENTINV, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=INVINFO, SEGTYPE=S1, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
   DESCRIPTION='Product Number'
   DESC='Product Number',
   DESC_ES='Numero de Producto',
   DESC_FR='Nombre de Produit', $
  FIELD=PRODNAME, ALIAS=PNAME, FORMAT=A30,
   WITHIN=PRODCAT,
   DESCRIPTION='Product Name'
   DESC_FR='Nom de Produit',
   DESC_ES='Nombre de Producto', $
  FIELD=QTY_IN_STOCK, ALIAS=QIS, FORMAT=I7,
   DESCRIPTION='Quantity In Stock', $
  FIELD=PRICE, ALIAS=RETAIL, FORMAT=D10.2,
   TITLE='Price:',
   DESCRIPTION=Price, $
```

Example: Using Multilingual Titles in a Request

The following Master File for the CENTINV data source specifies French titles (TITLE_FR) and Spanish titles (TITLE_ES) as well as default titles (TITLE) for the PROD_NUM and PRODNAME fields:

```
FILE=CENTINV, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=INVINFO, SEGTYPE=S1, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
   TITLE='Product,Number:',
   TITLE_FR='Nombre,de Produit:',
   TITLE_ES='Numero,de Producto:',
   DESCRIPTION='Product Number', $
  FIELD=PRODNAME, ALIAS=PNAME, FORMAT=A30,
   WITHIN=PRODCAT,
   TITLE='Product,Name:',
   TITLE_FR='Nom,de Produit:',
   TITLE_ES='Nombre,de Producto:'
   DESCRIPTION='Product Name', $
  FIELD=QTY_IN_STOCK, ALIAS=QIS, FORMAT=I7,
   TITLE='Quantity,In Stock:',
   DESCRIPTION='Quantity In Stock', $
  FIELD=PRICE, ALIAS=RETAIL, FORMAT=D10.2,
   TITLE='Price:',
   DESCRIPTION=Price, $
```

The default language is English and, by default, SET TITLES=ON. Therefore, the following request, uses the TITLE attributes to produce column headings that are all in English:

```
TABLE FILE CENTINV
PRINT PROD_NUM PRODNAME PRICE
WHERE PRICE LT 200
END
```

The output is:

```
Product   Product
Number:   Name:                                Price:
-------   -------                              ------
1004      2 Hd VCR LCD Menu                    179.00
1008      DVD Upgrade Unit for Cent. VCR       199.00
1026      AR3 35MM Camera 10 X                 129.00
1028      AR2 35MM Camera 8 X                  109.00
1030      QX Portable CD Player                169.00
1032      R5 Micro Digital Tape Recorder        89.00
```

Now, issue the following command to set the language to Spanish and run the same request:

```
SET LANG = SPA
```

The output now displays column headings from the TITLE_ES attributes where they exist (Product Number and Product Name). Where no Spanish title is specified (the Price field), the column heading in the TITLE attribute appears:

```
Numero        Nombre
de Producto:  de Producto:                          Price:
------------  ------------                          ------
1004          2 Hd VCR LCD Menu                     179.00
1008          DVD Upgrade Unit for Cent. VCR        199.00
1026          AR3 35MM Camera 10 X                  129.00
1028          AR2 35MM Camera 8 X                   109.00
1030          QX Portable CD Player                 169.00
1032          R5 Micro Digital Tape Recorder         89.00
```

Describing a Virtual Field: DEFINE

In this section:

Using a Virtual Field

Using Date System Amper Variables in Master File DEFINEs

How to:

Define a Virtual Field

Reference:

Usage Notes for Virtual Fields in a Master File

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

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

Syntax: How to Define a Virtual Field

```
DEFINE fieldname/format [REDEFINES fieldname2] = expression;
                  [, attribute2, ... ] $
```

where:

fieldname

Is the name of the virtual 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 93.

format

Is the field's format. It is specified in the same way as formats assigned using the USAGE attribute, which is described in *The Displayed Data Type: USAGE* on page 103. If you do not specify a format, it defaults to D12.2.

expression

Is a valid expression. 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 virtual field, it must include the ELSE phrase.

attribute2

The declaration for a virtual field can include additional optional attributes, such as TITLE and DESCRIPTION.

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 Virtual Fields in a Master File

Note the following rules when using DEFINE:

- **Alias.** DEFINE does not have an alias.
- **Changes.** You can change the virtual field's declaration at any time.
- A DEFINE FILE command takes precedence over a DEFINE in the Master with same name.
- If the expression used to derive the virtual field invokes a function, parameter numbers and types are not checked unless the USERFCHK parameter is set to FULL.

Using a Virtual Field

A DEFINE attribute cannot contain qualified field names on the left-hand side of the expression. Use the WITH phrase on the left-hand side to place the defined field in the same segment as any real field you choose. This will determine when the DEFINE expression will be evaluated.

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 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 virtual field that derives its value from fields in several different paths, you have to create it with a DEFINE FILE command using an alternate view 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 virtual field that is only used once, and you do not want to add a declaration for it to the Master File.

Virtual 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 virtual field cannot be used for cross-referencing in a join. It can, however, be used as a host field 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 message:

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

Using Date System Amper Variables in Master File DEFINEs

Example:

Using the Date Variable &DATE in a Master File DEFINE

Using the Date Variable &YYMD in a Master File DEFINE

Reference:

Messages for Date System Amper Variables in Master File DEFINEs

Master File DEFINE fields can use Dialogue Manager system date variables to capture the system date each time the Master File is parsed for use in a request.

The format of the returned value for each date variable is the format indicated in the variable name. For example, &DATEYYMD returns a date value with format YYMD. The exceptions are &DATE and &TOD, which return alphanumeric values and must be assigned to a field with an alphanumeric format. The variable names &DATE and &TOD must also be enclosed in single quotation marks in the DEFINE expression.

The variables supported for use in Master File DEFINEs are:

- &DATE
- &TOD
- &DATEMDY
- &DATEDMY
- &DATEYMD
- &DATEMDYY
- &DATEDMYY
- &DATEYYMD
- &DMY
- &YMD
- &MDY
- &YYMD
- &MDYY
- &DMYY

Note that all other reserved amper variables are not supported in Master Files.

Example: Using the Date Variable &DATE in a Master File DEFINE

The following version of the EMPLOYEE Master File has the DEFINE field named TDATE added to it. TDATE has format A12 and retrieves the value of &DATE, which returns an alphanumeric value and must be enclosed in single quotation marks:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,       ALIAS=EID,     FORMAT=A9,       $
 FIELDNAME=LAST_NAME,    ALIAS=LN,      FORMAT=A15,      $
 FIELDNAME=FIRST_NAME,   ALIAS=FN,      FORMAT=A10,      $
 FIELDNAME=HIRE_DATE,    ALIAS=HDT,     FORMAT=I6YMD,    $
 FIELDNAME=DEPARTMENT,   ALIAS=DPT,     FORMAT=A10,      $
 FIELDNAME=CURR_SAL,     ALIAS=CSAL,    FORMAT=D12.2M,   $
 FIELDNAME=CURR_JOBCODE, ALIAS=CJC,     FORMAT=A3,       $
 FIELDNAME=ED_HRS,       ALIAS=OJT,     FORMAT=F6.2,     $
DEFINE TDATE/A12   ='&DATE';, $
   .
   .
   .
```

The following request displays the value of TDATE:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE TDATE AS 'TODAY''S,DATE'
WHERE LAST_NAME EQ 'BANNING'
END
```

The output is:

```
                                        TODAY'S
LAST_NAME        FIRST_NAME  HIRE_DATE  DATE
---------        ----------  ---------  -------
BANNING          JOHN         82/08/01  05/11/04
```

Example: Using the Date Variable &YYMD in a Master File DEFINE

The following version of the EMPLOYEE Master File has the DEFINE field named TDATE added to it. TDATE has format YYMD and retrieves the value of &YYMD:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,        ALIAS=EID,     FORMAT=A9,        $
 FIELDNAME=LAST_NAME,     ALIAS=LN,      FORMAT=A15,       $
 FIELDNAME=FIRST_NAME,    ALIAS=FN,      FORMAT=A10,       $
 FIELDNAME=HIRE_DATE,     ALIAS=HDT,     FORMAT=I6YMD,     $
 FIELDNAME=DEPARTMENT,    ALIAS=DPT,     FORMAT=A10,       $
 FIELDNAME=CURR_SAL,      ALIAS=CSAL,    FORMAT=D12.2M,    $
 FIELDNAME=CURR_JOBCODE, ALIAS=CJC,      FORMAT=A3,        $
 FIELDNAME=ED_HRS,        ALIAS=OJT,     FORMAT=F6.2,      $
DEFINE TDATE/YYMD   = &YYMD ;, $
   .
   .
   .
```

The following request displays the value of TDATE:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE TDATE AS 'TODAY''S,DATE'
WHERE LAST_NAME EQ 'BANNING'
END
```

The output is:

```
                                        TODAY'S
LAST_NAME        FIRST_NAME  HIRE_DATE  DATE
---------        ----------  ---------  -------
BANNING          JOHN         82/08/01  2004/05/11
```

Reference: Messages for Date System Amper Variables in Master File DEFINEs

The following message appears if an attempt is made to use an unsupported amper variable in a Master File DEFINE:

```
(FOC104) DEFINE IN MASTER REFERS TO A FIELD OUTSIDE ITS SCOPE: var
```

Describing a Filter: FILTER

How to:

Declare a Filter in a Master File

Use a Master File Filter in a Request

Reference:

Usage Notes for Filters in a Master File

Example:

Defining and Using a Master File Filter

Boolean virtual fields (DEFINE fields that evaluate to TRUE or FALSE) can be used as record selection criteria. If the primary purpose of a virtual field is for use in record selection, you can clarify this purpose and organize virtual fields in the Master File by storing the expression using a FILTER declaration rather than a DEFINE. Filters offer the following features:

- They allow you to organize and store popular selection criteria in a Master File and reuse them in multiple requests and tools.
- For some data sources (such as VSAM and ISAM), certain filter expressions can be inserted inline into the WHERE or IF clause, enhancing optimization compared to a Boolean DEFINE.

Syntax: How to Declare a Filter in a Master File

```
FILTER  filtername = expression;
```

where:

filtername

Is the name assigned to the filter. The filter is internally assigned a format of I1, which cannot be changed.

expression

Is a logical expression that evaluates to TRUE (which assigns the value 1 to the filter field) or FALSE (which assigns the value 0 to the filter field). For any other type of expression, the field becomes a standard numeric virtual field in the Master File. Dialogue Manager variables (amper variables) can be used in the filter expression in same way they are used in standard Master File DEFINEs.

Syntax: How to Use a Master File Filter in a Request

```
TABLE FILE filename
    .
    .
    .
{WHERE|IF} expression_using_filters
```

where:

expression_using_filters

Is a logical expression that references a filter. In a WHERE phrase, the logical expression can reference one or more filters and/or virtual fields.

Reference: Usage Notes for Filters in a Master File

- The filter field name is internally assigned a format of I1 which cannot be changed.
- A filter can be used as a standard numeric virtual field anywhere in a report request, except that they are not supported in WHERE TOTAL tests.

Example: Defining and Using a Master File Filter

Consider the following filter declaration added to the MOVIES Master File:

```
FILTER G_RATING = RATING EQ 'G' OR 'PG'; $
```

The following request applies the G_RATING filter:

```
TABLE FILE MOVIES
HEADING CENTER
"Rating G and PG"
PRINT TITLE CATEGORY RATING
WHERE G_RATING
ON TABLE SET PAGE NOPAGE
ON TABLE SET GRID OFF
ON TABLE HOLD FORMAT HTML
ON TABLE SET STYLE *
type=report, style=bold, color=black, backcolor=yellow, $
type=data, backcolor=aqua, $
ENDSTYLE
END
```

The output is:

Rating G and PG		
TITLE	CATEGORY	RATING
JAWS	ACTION	PG
CABARET	MUSICALS	PG
BABETTE'S FEAST	FOREIGN	G
SHAGGY DOG, THE	CHILDREN	G
REAR WINDOW	MYSTERY	PG
VERTIGO	MYSTERY	PG
BACK TO THE FUTURE	COMEDY	PG
GONE WITH THE WIND	CLASSIC	G
AIRPLANE	COMEDY	PG
ALICE IN WONDERLAND	CHILDREN	G
ANNIE HALL	COMEDY	PG
FIDDLER ON THE ROOF	MUSICALS	G
BIG	COMEDY	PG
TOP GUN	ACTION	PG
FAMILY, THE	FOREIGN	PG
BAMBI	CHILDREN	G
DEATH IN VENICE	FOREIGN	PG

Describing a Calculated Value: COMPUTE

How to:

Include a COMPUTE Command in a Master File

Reference:

Usage Notes for COMPUTE in a Master File

Example:

Coding a COMPUTE in the Master File and Accessing the Computed Value

COMPUTE commands can be included in Master Files and referenced in subsequent TABLE requests, enabling you to build expressions once and use them in multiple requests.

Syntax: How to Include a COMPUTE Command in a Master File

```
COMPUTE fieldname/fmt=expression;
```

where:

fieldname
: Is name of the calculated field.

fmt
: Is the format and length of the calculated field.

expression
: Is the formula for calculating the value of the field.

Reference: Usage Notes for COMPUTE in a Master File

In all instances, COMPUTEs in the Master File have the same functionality and limitations as temporary COMPUTEs. Specifically, fields computed in the Master File must follow these rules:

- ❏ They cannot be used in JOIN, DEFINE, or ACROSS phrases, or with prefix operators.
- ❏ When used as selection criteria, syntax is either IF TOTAL field or WHERE TOTAL field.
- ❏ When used as sort fields, syntax is BY TOTAL COMPUTE field.
- ❏ To insert a calculated value into a heading or footing, you must reference it prior to the HEADING or FOOTING command.

Example: Coding a COMPUTE in the Master File and Accessing the Computed Value

Use standard COMPUTE syntax to add a calculated value to your Master File. You can then access the calculated value by referencing the computed fieldname in subsequent TABLE requests. When used as a verb object, as in the following example, the syntax is SUM (or PRINT) COMPUTE field.

The following is the SALESTES Master File (the SALES FILE modified with an embedded COMPUTE):

```
FILENAME=SALESTES, SUFFIX=FOC,
SEGNAME=STOR_SEG, SEGTYPE=S1,
   FIELDNAME=STORE_CODE,  ALIAS=SNO,  FORMAT=A3,   $
   FIELDNAME=CITY,        ALIAS=CTY,  FORMAT=A15,  $
   FIELDNAME=AREA,        ALIAS=LOC,  FORMAT=A1,   $

SEGNAME=DATE_SEG, PARENT=STOR_SEG, SEGTYPE=SH1,
   FIELDNAME=DATE,        ALIAS=DTE,  FORMAT=A4MD, $

SEGNAME=PRODUCT, PARENT=DATE_SEG, SEGTYPE=S1,
   FIELDNAME=PROD_CODE,     ALIAS=PCODE,   FORMAT=A3,    FIELDTYPE=I, $
   FIELDNAME=UNIT_SOLD,     ALIAS=SOLD,    FORMAT=I5,    $
   FIELDNAME=RETAIL_PRICE,  ALIAS=RP,      FORMAT=D5.2M, $
   FIELDNAME=DELIVER_AMT,   ALIAS=SHIP,    FORMAT=I5,    $
   FIELDNAME=OPENING_AMT,   ALIAS=INV,     FORMAT=I5,    $
   FIELDNAME=RETURNS,       ALIAS=RTN,     FORMAT=I3,    MISSING=ON, $
   FIELDNAME=DAMAGED,       ALIAS=BAD,     FORMAT=I3,    MISSING=ON, $

   COMPUTE REVENUE/D12.2M=UNIT_SOLD*RETAIL_PRICE;
```

In the TABLE request, computed field, REVENUE, is a verb object of SUM.

```
TABLE FILE SALESTES
HEADING CENTER
"NEW YORK PROFIT REPORT"
" "
SUM UNIT_SOLD AS 'UNITS,SOLD' RETAIL_PRICE AS 'RETAIL_PRICE'
COMPUTE REVENUE;
BY PROD_CODE AS 'PROD,CODE'
WHERE CITY EQ 'NEW YORK'
END
```

The output is:

```
          NEW YORK PROFIT REPORT

PROD  UNITS
CODE  SOLD   RETAIL_PRICE          REVENUE
----  ----   ------------          -------
B10     30          $.85           $25.50
B17     20         $1.89           $37.80
B20     15         $1.99           $29.85
C17     12         $2.09           $25.08
D12     20         $2.09           $41.80
E1      30          $.89           $26.70
E3      35         $1.09           $38.15
```

5 Describing a Sequential, VSAM, or ISAM Data Source

You can describe and report from sequential, VSAM, and ISAM data sources.

In a sequential data source, records are stored and retrieved in the same order as they are entered.

With VSAM and ISAM data sources, a new element is introduced: the key or group key. A group key consists of one or more fields, and can be used to identify the various record types in the data source. In the Master File representation of a data source with different record types, each record type is assigned its own segment.

For VSAM and ISAM data sources, you must allocate (or DLBL in DOS/VSE and VM) the Master File name to the CLUSTER component of the data source.

See Appendix C, *User Exits for a Non-FOCUS Data Source* for details. For information about updating VSAM data sources, see the *VSAM Write Data Adapter User's Manual*.

Topics:

Sequential Data Source Formats

In this section:

Sequential data sources formatted in the following ways are recognized:

- Fixed-format, in which each field occupies a predefined position in the record.
- Comma- or tab-delimited, in which fields can occupy any position in a record and are separated by a comma or a tab, respectively. Ending a line terminates the record.

  Free-format is a type of comma-delimited data source in which a record can span multiple lines and is terminated by a comma-dollar sign (,$) character combination.
- Token-delimited, in which the delimiter can be any combination of characters. For information on describing token-delimited files, see *Describing a Token-Delimited Data Source* on page 237.

You can describe two types of sequential data sources:

- **Simple.** This is the most basic type, consisting of only one segment. It is supported in all formats.
- **Complex.** This is a multi-segment data source. The descendant segments exist in the data source 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 Is a Fixed-Format Data Source?

Fixed-format data sources are sequential data sources in which each field occupies a predefined 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 data source can be described by providing just field declarations. For example, suppose you have a data source 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 data source might be described with 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).
- ALIAS=, USAGE=, and ACTUAL= may be omitted as identifiers, since they are positional attributes following FIELDNAME.
- If you omit the optional ALIAS, its absence must be signaled by a second comma between FIELDNAME and USAGE (FIELDNAME=PUBNO,,A10,A10,$).
- Both the USAGE and the ACTUAL attributes must be included. Failure to specify both is a common cause of errors in describing non-FOCUS data sources (FOCUS data sources do not have ACTUAL attributes).
- Each declaration can span multiple lines and must be terminated with a comma followed by a dollar sign (,$). Typically, the LRECL for a Master File is 80 and the RECFM is F. Support also exists for LRECL up to 32K and RECFM V.
- When using Maintain to read a fixed-format data source, the record length as described in the Master File may not exceed the actual length of the data record (the LRECL value).

You must describe the entire record. The values for field name and alias can be omitted for fields that do not need to be accessed. This is significant when using existing data sources, because they frequently contain information that you do not need for your requests. You describe only the fields to include in your reports or calculations, and use filler fields to represent the rest of the logical record length (LRECL) of the data source.

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 4095 characters. If you need to describe larger areas, use several filler fields together:

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

The field name FILLER is no different than any other field name. To prevent access to the data in the field, you can use a blank field name. For example:

```
FIELDNAME =,,A200,A200,$
```

It is recommended that you include file and segment attributes, even for simple data sources, to complete your documentation. The example below shows the Master File for the library data source 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 Is a Comma or Tab-Delimited Data Source?

Reference:

Accessing SUFFIX=COM Data Sources

Accessing SUFFIX=COMT Data Sources

Accessing SUFFIX=TAB Data Sources

Accessing SUFFIX=TABT Data Sources

Master Files for comma- and tab-delimited sequential data sources can have SUFFIX values of COM, COMT, TAB, or TABT. Comma-delimited data sources are sequential data sources in which field values are separated by commas. Tab-delimited data sources are sequential data sources in which field values are separated by tabs.

You can use the SET HNODATA command to specify how to propagate missing data to these data sources when you create them using the HOLD command. For more information, see the *Developing Applications* manual.

For the SUFFIX values of COM, COMT, TAB and TABT, if the data is numeric and has a zoned format (ACTUAL=Z*n*), the data must be unsigned (can not contain a positive or negative value).

Reference: Accessing SUFFIX=COM Data Sources

A Master File containing the attribute SUFFIX=COM can be used to access two styles of comma-delimited sequential data sources:

- Free-format style is described in *What Is a Free-Format Data Source?* on page 187. Character values are not enclosed in double quotation marks, and the comma-dollar sign character combination terminates the record. With this style of comma-delimited data source, records can span multiple lines. A field that contains a comma as a character must be enclosed within single quotation marks.

- The second style is consistent with the current industry standard for comma-delimited data sources. Character values are enclosed in double quotation marks, and the cr/lf (carriage-return, line-feed) character combination terminates the record, although the comma-dollar sign sequence is also accepted. In addition, each input record must be completely contained on a single input line. A double quotation mark within a field is identified by two consecutive double quotation marks.

  Note that the setting PCOMMA=ON is required in conjunction with the SUFFIX=COM Master File when accessing this type of data source in order to interpret the double quotation marks around character values correctly. Without this setting, the double quotation marks are considered characters within the field, not delimiters enclosing the field values.

Reference: Accessing SUFFIX=COMT Data Sources

A Master File containing the attribute SUFFIX=COMT can be used to access comma-delimited sequential data sources in which all of the following conditions are met:

- The first record of the data source contains column titles. This record is ignored when the data source is accessed in a request.
- Character values are enclosed in double quotation marks. A double quotation mark within a field is identified by two consecutive double quotation marks.
- Each record is completely contained on one line and terminated with the cr/lf character combination.

Reference: Accessing SUFFIX=TAB Data Sources

A Master File containing the attribute SUFFIX=TAB can be used to access tab-delimited sequential data sources in which all of the following conditions are met:

- Character values are not enclosed in double quotation marks.
- Each record is completely contained on one line and terminated with the cr/lf character combination.

Reference: Accessing SUFFIX=TABT Data Sources

A Master File containing the attribute SUFFIX=TABT can be used to access tab-delimited sequential data sources in which all of the following conditions are met:

- ❏ The first record of the data source contains column titles. This record is ignored when the data source is accessed in a request.
- ❏ Character values are not enclosed in double quotation marks.
- ❏ Each record is completely contained on one line and terminated with the cr/lf character combination.

What Is a Free-Format Data Source?

A common type of external structure is a comma-delimited sequential data source. These data sources 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 zeroes to fixed field lengths. Comma-delimited data sources must be stored as physical sequential data sources.

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

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

Note: Free-format comma-delimited data sources do not have character fields enclosed in double quotation marks, and use the comma-dollar sign character combination to terminate the record.

You can use the system editor to change values, add new records, and delete records. Since the number of data fields on a line varies 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 data sources. 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 a Free-Format Data Source

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

A logical record may occupy as many lines in the data source 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 the following 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.
- If a field name is not mentioned, it inherits its value from the prior record.

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 for a Sequential Data Source

Most standard Master File attributes—those described in Chapter 2, *Identifying a Data Source*, Chapter 3, *Describing a Group of Fields*, and Chapter 4, *Describing an Individual Field*—are used with sequential data sources in the standard way.

- **SEGTYPE.** The SEGTYPE attribute is ignored with free-format data sources.

  The SEGTYPE value for fixed-format data sources has a default of S0. However, if you use keyed retrieval, the SEGTYPE value depends on the number of keys and sort sequence. See Chapter 6, *Describing a FOCUS Data Source*, for a description of the SEGTYPE attribute. For a description of keyed retrieval from fixed-format data sources, see the *Creating Reports* manual.

- **ACTUAL.** The ACTUAL values for sequential data sources are described in Chapter 4, *Describing an Individual Field*.

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

Standard Master File Attributes for a VSAM or ISAM Data Source

In this section:

Most standard Master File attributes—those described in Chapter 2, *Identifying a Data Source*, Chapter 3, *Describing a Group of Fields*, and Chapter 4, *Describing an Individual Field*—are used with VSAM and ISAM data sources in the standard way.

- **SUFFIX.** The SUFFIX attribute in the file declaration for these data sources has the value VSAM or ISAM.
- **SEGNAME.** The SEGNAME attribute of the first or root segment in a Master File for a VSAM or ISAM data source must be ROOT. The remaining segments can have any valid segment name.

  The only exception involves unrelated RECTYPEs, where the root 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 that 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.** The SEGTYPE attribute should be S0 for VSAM data sources. (For a general description of the SEGTYPE attribute, see Chapter 3, *Describing a Group of Fields*.)
- **GROUP.** The keys of a VSAM or ISAM data source are defined in the segment declarations as GROUPs consisting of one or more fields.

Describing a Group Field With Formats

How to:

Describe a VSAM Group Field With Formats

Example:

Describing a VSAM Group Field With Formats

Describing a VSAM Group Field With Multiple Formats

Accessing a Group Field With Multiple Formats

A single-segment data source may have only one key field, 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 that you have a series of three fields for an employee: last name; first name; and middle initial. 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.

The GROUP declaration USAGE attribute specifies how many positions to use to describe the key in a VSAM KSDS data source. If a Master File does not completely describe the full key at least once, the following warning message appears:

```
(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 data source, 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.

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 because regardless of the data types, the group will be displayed as alphanumeric. 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 of type P that are 8 bytes can have a USAGE of P15.x or P16 (sign and decimal for a total of 15 digits). Fields that are 16 bytes have a USAGE of P16.x, 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.

Note:

- Since all group fields must be defined in alphanumeric format, those that include numeric component fields should not be used as verb objects in a report request.
- The MISSING attribute is not supported on the group field, but is supported on the individual fields comprising the group.

Syntax: How to Describe a VSAM Group Field With Formats

```
GROUP = keyname, ALIAS = KEY, USAGE = Ann, ACTUAL = Ann ,$
```

where:

keyname

Can have up to 66 characters.

Example: Describing a VSAM Group Field With Formats

In the library data source, the first field, PUBNO, can be 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. They can be described as a group key, consisting of a separate field for each element if the data source is a VSAM data structure.

The Master File looks 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   ,$
```

Example: Describing a VSAM Group Field With Multiple Formats

```
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 display lengths of the USAGE attributes for the subordinate fields total 17. However, the length of the group key USAGE attribute is found by 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.

Example: Accessing a Group Field With Multiple Formats

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. Because FIELD2 is a numeric field, you cannot specify the IF criteria as follows:

```
IF GRPB EQ 'A 0334BB'
```

You can eliminate this error by using a slash (/) to separate the components of the group key:

```
IF GRPB EQ 'A/334/BB'
```

Note: Blanks and leading zeros are assumed where needed to fill out the key.

Describing a Group Field as a Set of Elements

How to:

Describe a GROUP Field as a Set of Elements

Reference:

Usage Notes for Group Elements

Example:

Declaring a GROUP With ELEMENTS

A GROUP declaration in a Master File describes several fields as a single entity. One use of a group is to describe group keys in a VSAM data source. Sometimes referring to several fields by one group name facilitates ease of reporting.

Traditionally, when describing a GROUP field, you had to take account of the fact that while the USAGE and ACTUAL format for the GROUP field are both alphanumeric, the length portion of the USAGE format for the group had to be calculated as the sum of the component lengths, where each integer or single precision field counted as 4 bytes, each double precision field as 8 bytes, and each packed field counted as either 8 or 16 bytes depending on its size.

To avoid the need to calculate these lengths, you can use the GROUP ELEMENTS option, which describes a group as a set of elements without USAGE and ACTUAL formats.

Syntax: How to Describe a GROUP Field as a Set of Elements

```
GROUP=group1, ALIAS=g1alias,ELEMENTS=n1,$
   FIELDNAME=field11, ALIAS=alias11, USAGE=ufmt11, ACTUAL=afmt11, $
   .
   .
   .
   FIELDNAME=field1h, ALIAS=alias1h, USAGE=ufmt1h, ACTUAL=afmt1h, $
GROUP=group2,ALIAS=g2alias,ELEMENTS=n2,$
   FIELDNAME=field21, ALIAS=alias21, USAGE=ufmt21, ACTUAL=afmt21, $
   .
   .
   .
   FIELDNAME=field2k, ALIAS=alias2k, USAGE=ufmt2k, ACTUAL=afmt2k, $
```

where:

group1, group2

Are valid names assigned to a group of fields. The rules for acceptable group names are the same as the rules for acceptable field names.

n1, n2

Are the number of elements (fields and/or groups) that compose the group. If a group is defined within another group, the subgroup (with all of its elements) counts as one element of the parent group.

field11, field2k

Are valid field names.

alias11, alias2k

Are valid alias names.

ufmt11, ufmt2k

Are USAGE formats for each field.

afmt11, afmt2k

Are ACTUAL formats for each field.

Reference: Usage Notes for Group Elements

- To use the ELEMENTS attribute, the GROUP field declaration should specify only a group name and number of elements.
  - If a group declaration specifies USAGE and ACTUAL *without* the ELEMENTS attribute, the USAGE and ACTUAL are accepted as specified, even if incorrect.
  - If a group declaration specifies USAGE and ACTUAL *with* the ELEMENTS attribute, the ELEMENTS attribute takes precedence.
- Each subgroup counts as one element. Its individual fields and subgroups do not count in the number of elements of the parent group.

Example: Declaring a GROUP With ELEMENTS

In the following Master File, GRP2 consists of two elements, fields FIELDA and FIELDB. GRP1 consists of two elements, GRP2 and field FIELDC. Field FIELDD is not part of a group:

```
FILENAME=XYZ      , SUFFIX=FIX      , $
  SEGMENT=XYZ, SEGTYPE=S2, $
GROUP=GRP1,ALIAS=CCR,ELEMENTS=2,$
   GROUP=GRP2,ALIAS=CC,ELEMENTS=2,$
    FIELDNAME=FIELDA, ALIAS=E01, USAGE=A10, ACTUAL=A10, $
    FIELDNAME=FIELDB, ALIAS=E02, USAGE=A16, ACTUAL=A16, $
    FIELDNAME=FIELDC, ALIAS=E03, USAGE=P27, ACTUAL=A07, $
    FIELDNAME=FIELDD, ALIAS=E04, USAGE=D7, ACTUAL=A07, $
```

The following chart shows the offsets and formats of these fields.

Field Number	Field Name	Offset	USAGE	ACTUAL
1	GRP1	0	A42 - Supports 16 characters for FIELDC (P27)	A33
2	GRP2	0	A26	A26
3	FIELDA	0	A10	A10
4	FIELDB	10	A16	A16
5	FIELDC	26	P27	A7
6	FIELDD	42	D7	A7

Note that the display characteristics of the group have not changed. The mixed format group GRP1 will still display as all alphanumeric.

Describing a Multiply Occurring Field in a Free-Format Data Source

Example:

Describing a Multiply Occurring Field in a Free-Format Data Source

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

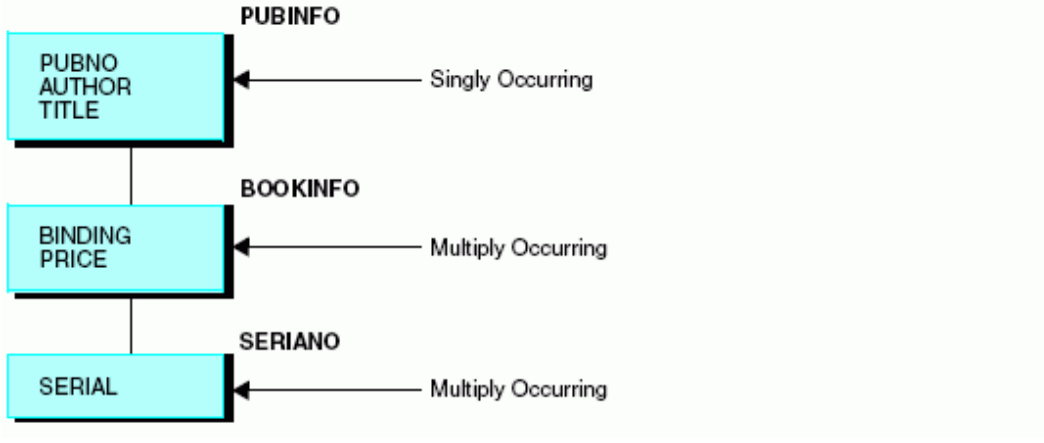
Example: Describing a Multiply Occurring Field in a Free-Format Data Source

Consider the example of a library data source. 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 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:



Describe this data source as follows:

```
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 a Multiply Occurring Field in a Fixed-Format, VSAM, or ISAM Data Source

In this section:

Fixed-format sequential, VSAM, or ISAM data sources 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 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.

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.

Using the OCCURS Attribute

How to:

Specify a Repeating Field

Example:

Using the OCCURS Attribute

The OCCURS attribute is an optional segment attribute used to describe records containing repeating fields or groups of fields. Define such records 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. Describe each of these sets of fields as a separate segment in your data source description. Sets of repeating fields can be divided into two basic types: parallel and nested.

Syntax: How to Specify a Repeating Field

```
OCCURS = occurstype
```

Possible values for *occurstype* are:

n

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

fieldname

Names a field in the parent segment whose integer value contains 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 (for example, if the field lengths for the segment add up to 40, and 120 characters are read in, it means there are three occurrences).

Place the OCCURS attribute in your segment declaration after the PARENT attribute.

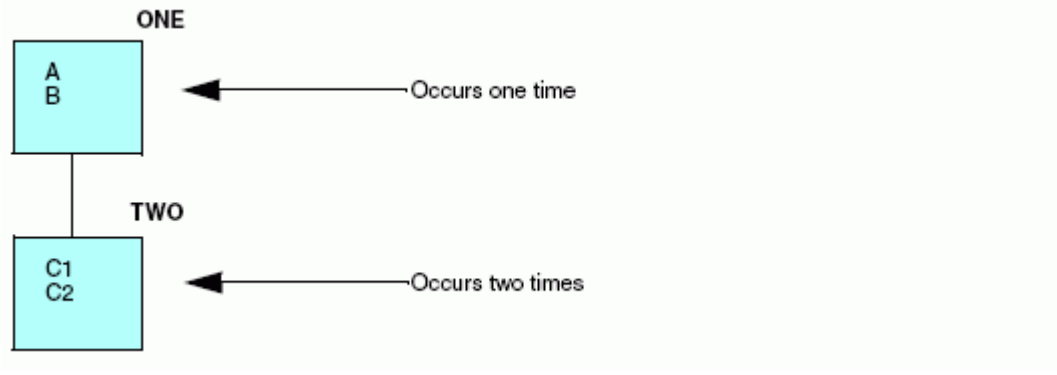
When different types of records are combined in one data source, each record type can contain only one segment defined as OCCURS=VARIABLE. It may have OCCURS descendants (if it contains a nested group), but it may not be followed by any other segment with the same parent—that is, there can be no other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

Example: Using the OCCURS 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 source, you place fields A and B in the root segment, and fields C1 and C2 in a descendant segment, as shown here:



Describe this data source as follows:

```
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, $
```

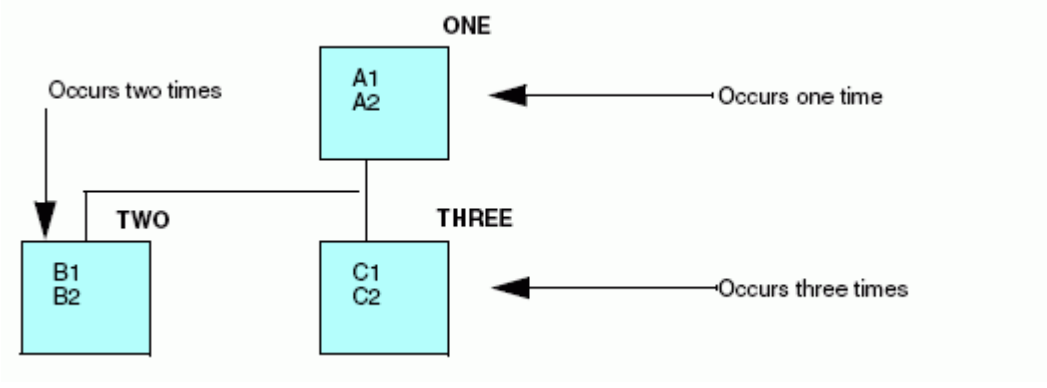
Describing a Parallel Set of Repeating Fields

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 data source description as children of the same parent, the segment that contains fields A1 and A2.

The following data structure reflects their relationship:



Describing a Nested Set of Repeating Fields

Example:

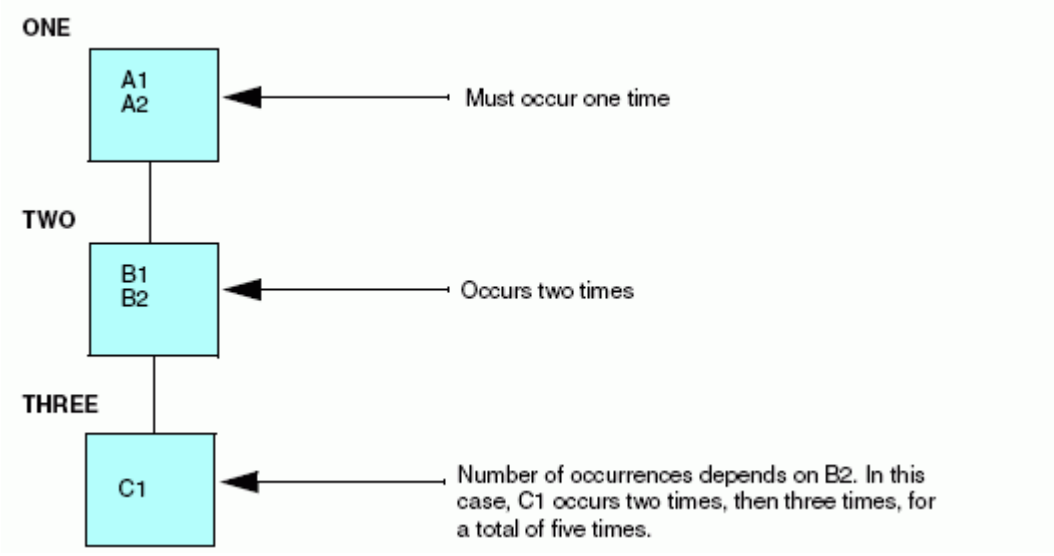
Describing Parallel and Nested Repeating Fields

Nested sets of repeating fields are those whose occurrence depends on one another in some way. 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. There is not a set of occurrences of C1 which is not preceded by an occurrence of fields B1 and B2. Fields B1, B2, and C1 are a nested set of repeating fields.

These repeating fields can be represented by the following data structure:



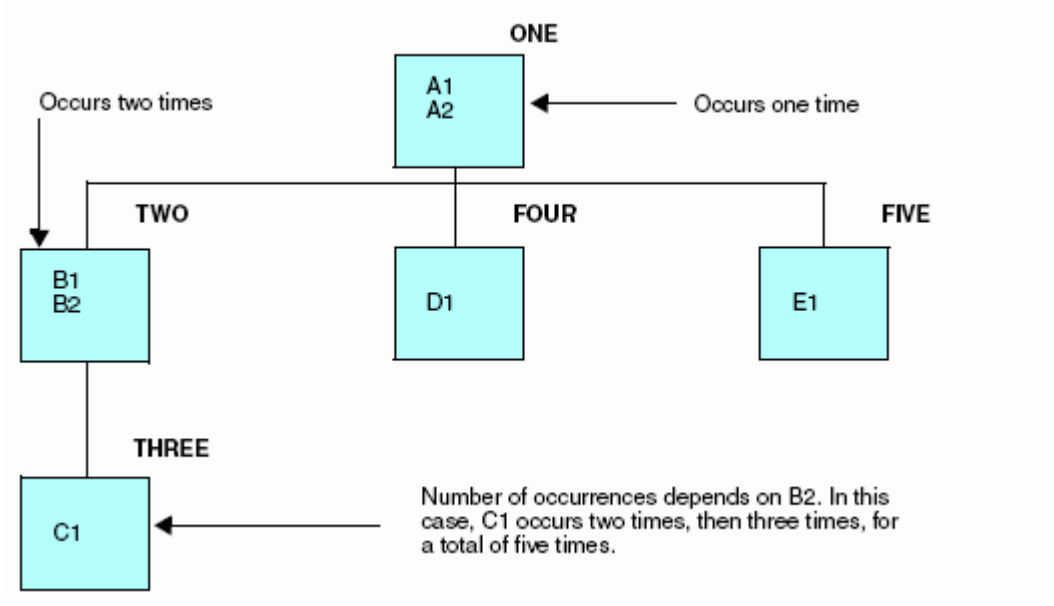
Since field C1 repeats with relation to fields B1 and B2, which repeat in relation to fields A1 and A2, field C1 is described as a separate, descendant segment of Segment TWO, which is in turn a descendant of Segment ONE.

Example: Describing Parallel and Nested Repeating Fields

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

A 1	A 2	B 1	B 2	C 1	C 1	C 1	B 1	B 2	C 1	C 1	C 1	C 1	D 1	D 1	E 1	E 1	E 1	E 1

It produces the following data structure:



Describe this data source as follows. 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 all the rest of the fields in the record (all those to the right of the first occurrence of E1) are read as recurrences of field E1. To ensure that data in the record is interpreted unambiguously, a segment defined as OCCURS=VARIABLE must be at the end of the record. In a data structure diagram, it is the rightmost segment. Note that there can be only one segment defined as OCCURS=VARIABLE for each record type.

Using the POSITION Attribute

How to:

Specify the Position of a Repeating Field

Example:

Specifying the Position of a Repeating Field

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 source 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 one or more alphanumeric fields that appear where the multiply occurring fields appear in the record. The alphanumeric field may be 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 field in the parent segment is 32 characters.

You can also use the POSITION attribute to redescribe fields with SEGTYPE=U. See *Redefining a Field in a Non-FOCUS Data Source* on page 207.

Syntax: How to Specify the Position of a Repeating Field

The POSITION attribute is described in the child segment. It gives the name of the field in the parent segment that specifies the starting position and overall length 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 starting position of the multiple field occurrences.

Example: Specifying the Position of a Repeating Field

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, you specify a field or fields that occupy the position of the four Q1 fields in the record. You then assign the actual Q1 fields to a multiply occurring descendant segment. The POSITION attribute, specified in the descendant segment, gives the name of the field in the parent segment that identifies the starting position and overall length of the Q fields.

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 fields is longer than 4095, 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 4095 bytes.

Notice that this structure works only 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=*fieldname* or OCCURS=VARIABLE does not work.

Specifying the ORDER Field

How to:

Specify the Sequence of a Repeating 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.

To associate a sequence number with each occurrence of the field, you may define an internal counter field in any OCCURS segment. A value is automatically supplied that defines the sequence number of each repeating group.

Syntax: How to Specify the Sequence of a Repeating Field

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 I*n*, 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. It does not alter the logical record length (LRECL) of the data source being accessed.

Redefining a Field in a Non-FOCUS Data Source

How to:

Redefine a Field

Example:

Redefining a VSAM Structure

Reference:

Special Considerations for Redefining a Field

Redefining record fields in non-FOCUS data sources is supported. This enables you to describe a field 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.

The redefined fields can have any user-defined name.

Syntax: How to Redefine a Field

```
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 first field being redefined. Using the unique segment with redefined fields helps avoid problems with multipath reporting.

A one-to-one relationship forms 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 ,$
```

Reference: Special Considerations for Redefining a Field

- ❏ 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.
- ❏ For non-alphanumeric fields, you must know your data. Attempts to print numeric fields that contain alphanumeric data produce data exceptions or errors converting values. It is recommended that the first definition always be alphanumeric to avoid conversion errors.
- ❏ More than one field can be redefined in a segment.
- ❏ Redefines are supported only for IDMS, IMS, VSAM, DB2, and FIX data sources.

Extra-Large Record Length Support

How to:

Define the Maximum Record Length

MAXLRECL indicates the largest actual file record length that FOCUS can read. The limit for MAXLRECL is 65536 bytes, allowing the user to read a record twice as large as the length of the internal matrix, which is limited to 32K.

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

Syntax: How to Define the Maximum Record Length

```
SET MAXLRECL = nnnnn
```

where:

nnnnn

Is up to 65536.

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

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

Describing Multiple Record Types

In this section:

Fixed-format sequential, VSAM, and ISAM data sources can contain more than one type of record. When they do, they can be structured in one of two ways:

- A positional 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 positional 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 by defining the parent as the root, and the detail segments as descendants.

  Some VSAM and ISAM data sources are structured so that descendant records relate to each other through concatenating key fields. That is, the key fields 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 using a GROUP declaration. Each segment's GROUP key fields consist of the renamed key fields from the parent segment plus the unique key field from the child record.

- The records have no meaningful positional relationship, and records of varying types exist independently of each other in the data source.

  If the records have no meaningful positional relationship, you have to provide some means for interpreting the type of record that has been read. Do this by creating a dummy root segment for the records.

In order to describe sequential data sources 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 source is described as a multi-segment structure.

Key-sequenced VSAM and complex ISAM data sources also use the RECTYPE attribute to distinguish various record types within the data source.

A parent does not always share its RECTYPE with its descendants. It may share some other identifying piece of information, such as the PUBNO in the 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 VSAM or ISAM data sources with group keys, the RECTYPE field can be part of the segment's group key only when it belongs to a segment that has no descendants, or to a segment whose descendants are described with an OCCURS attribute. In *Describing VSAM Positionally Related Records* on page 216, the RECTYPE field is added to the group key in the SERIANO segment, the lowest descendant segment in the chain.

Describing a RECTYPE Field

How to:

Specify a Record Type Field

Example:

Specifying the RECTYPE Field

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

Another technique for redefining the parts of records is to use the MAPFIELD and MAPVALUE attributes described in *Describing a Repeating Group Using MAPFIELD* on page 230.

Syntax: How to Specify a Record Type Field

The RECTYPE field must fall in the same physical location of each record in the data source, or the record is ignored. The syntax to describe the RECTYPE field is

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

where:

value

Is the record type in alphanumeric format, if an ACCEPT list is not specified. If there is an ACCEPT list, this can be any value.

format

Is the data type of the field. In addition to RECTYPE fields in alphanumeric format, RECTYPE fields in packed and integer formats (formats P and I) are supported. Possible values are:

`An` (where *n* is 1-4095) indicates character data, including letters, digits, and other characters.

`In` indicates ACTUAL (internal) format binary integers:

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

The USAGE format can be I1 through I9, depending on the magnitude of the ACTUAL format.

`Pn` (where *n* is 1-16) indicates packed decimal ACTUAL (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. 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 P*m.n*, where *m* is the total number of digits, and *n* is the number of decimal places. Thus, P11.1 means an eleven-digit number with one decimal place.

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. Separate each item in the list with 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: Specifying the RECTYPE Field

The following field description is of a one-byte packed RECTYPE field containing the value 1:

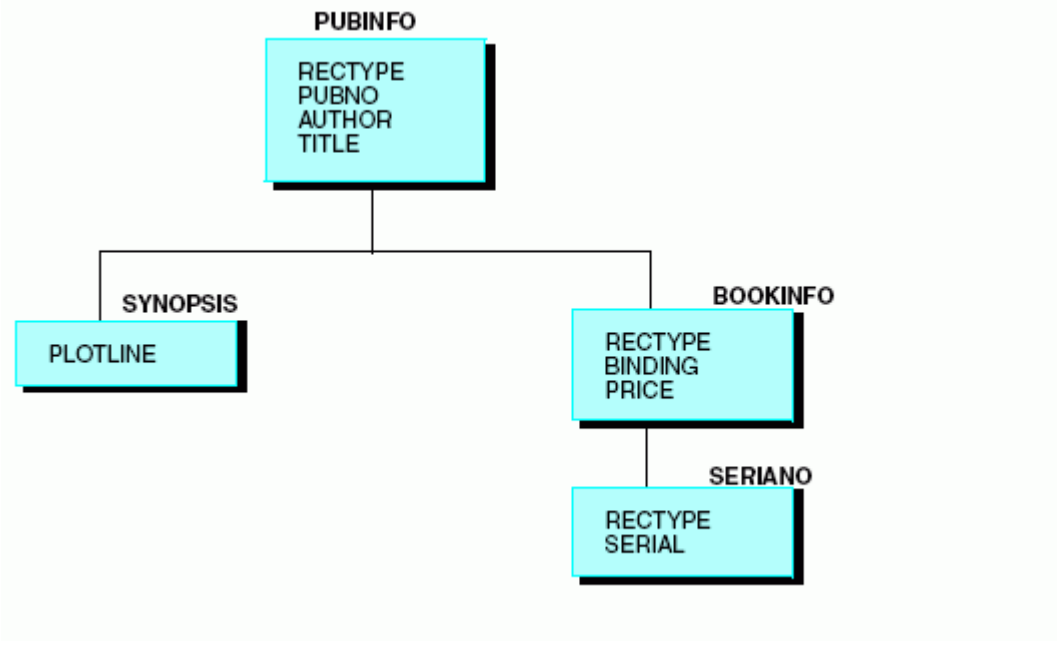
```
FIELD = RECTYPE, ALIAS = 1, USAGE = P1, ACTUAL = P1, $
```

The following field description is of a three-byte alphanumeric RECTYPE field containing the value A34:

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

Describing Positionally Related Records

The following diagram shows a more complex version of the library data source:



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. All of this information is assigned to the root segment in the Master File. The synopsis is common to all copies of a given book, but in this data source it is 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 in the diagram to the right of the OCCURS=VARIABLE segment, OCCURS=VARIABLE is the rightmost 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 structure. Note also that the OCCURS=VARIABLE segment does not have a RECTYPE. This is because it 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 has its own identifying serial number, which is 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.

Use the following Master File to describe this data source:

```
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 OCCURS, contains a field named RECTYPE and that the ALIAS for the field contains a unique value for each segment (P, B, and S). If there is a record in this data source with a RECTYPE other than P, B, or S, the record is ignored. The RECTYPE field must fall in the same physical location in each record.

Ordering of Records in the Data Source

Example:

Describing VSAM Positionally Related Records

Physical order determines parent/child relationships in sequential records. Every parent record does not need descendants. 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 example in *Describing Positionally Related Records* on page 214, if the first record in the data source is not a PUBINFO record, the record is considered to be a child without a parent. Any information allotted to the SYNOPSIS segment appears 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 if it has no parent.

Example: Describing VSAM Positionally Related Records

Consider the following VSAM data source 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.

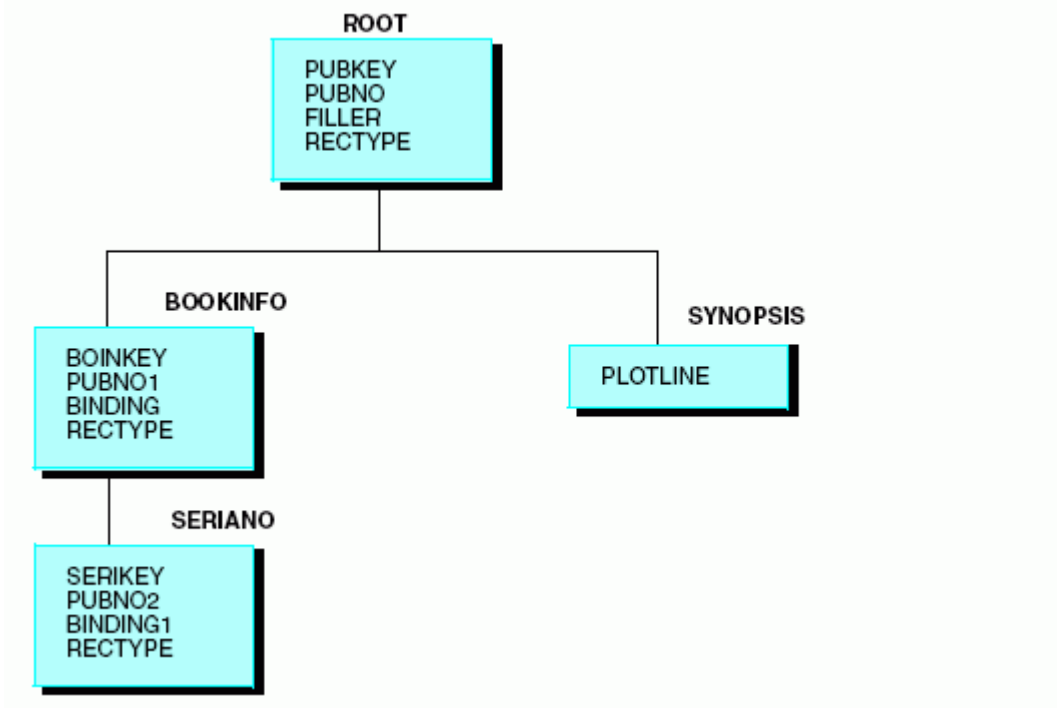
```
FILENAME = LIBRARY6, SUFFIX = VSAM,$
 SEGNAME = ROOT, SEGTYPE = S0,$
  GROUP=PUBKEY         ,ALIAS=KEY    ,USAGE=A10    ,ACTUAL=A10   ,$
   FIELDNAME=PUBNO     ,ALIAS=PN     ,USAGE=A10    ,ACTUAL=A10   ,$
  FIELDNAME=FILLER     ,ALIAS=       ,USAGE=A1     ,ACTUAL=A1    ,$
  FIELDNAME=RECTYPE    ,ALIAS=1      ,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 segments, 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. The length of the key increases as you traverse the structure.

Note: Each segment's key describes as much of the true key as needed to find the next instance of that segment.

In the sample data source, the repetition of the publisher's number as PUBNO1 and PUBNO2 in the descendant segments interrelates the three types of records.

The data source can be diagrammed as the following structure:



A typical query may 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, retrieval can occur quickly, and the processing continues. To further speed retrieval, add search criteria based on the BINDING field, which is also part of the key.

Describing Unrelated Records

Example:

Describing Unrelated Records Using a Dummy Root Segment

Describing a VSAM Data Source With Unrelated Records

Describing a Key and a Record Type for a VSAM Data Source With Unrelated Records

Some VSAM and ISAM data sources do not have records that are related to one another. That is, the VSAM or ISAM key of one record type is independent of the keys of other record types. To describe data sources with unrelated records, define a dummy root segment for the record types. 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 a blank name and alias.
- The USAGE and ACTUAL attributes must both be A1.

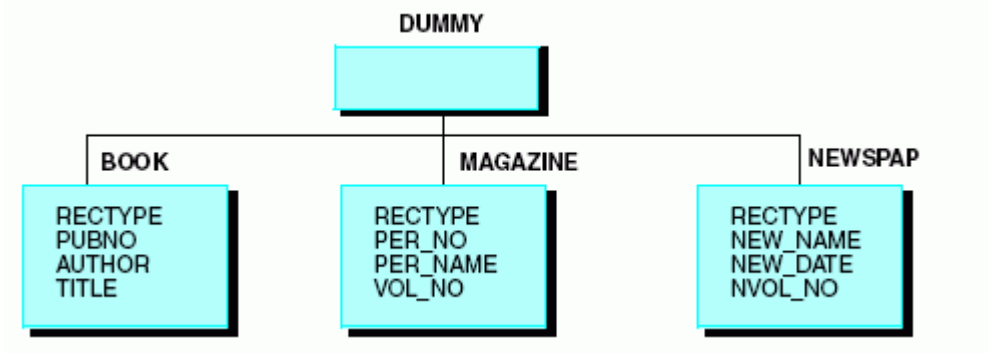
All other non-repeating segments must point to the dummy root as their parent. Except for the root, all non-repeating segments must have a RECTYPE and a PARENT attribute and describe the full VSAM/ISAM key. If the data source does not have a key, the group should not be described. RECTYPEs may be anywhere in the record.

Example: Describing Unrelated Records Using a Dummy Root Segment

The library data source has three types of records: book information, magazine information, and newspaper information. Since these three record types have nothing in common, they cannot be described as parent records followed by detail records.

The data source can look like this:

A structure such as the following can also describe this data source:



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   ,$
```

Example: Describing a VSAM Data Source With Unrelated Records

Consider another VSAM data source containing information on the library. This data source 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

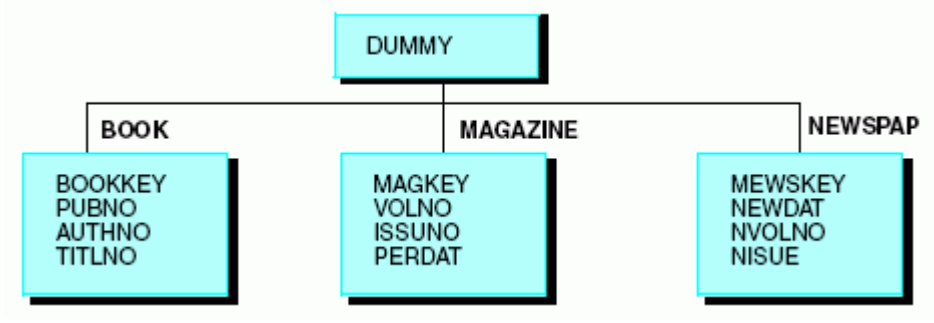
Note the difference between the use of the RECTYPE here and its use when the records are positionally related. In this case, the codes are unrelated and the database designer has chosen to accumulate the records by type first (all the book information together, all the magazine information together, and all the newspaper information together), so the RECTYPE may be the initial 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 data source can be arbitrary.

Both types of file structure can be represented by the following:



Example: Describing a Key and a Record Type for a VSAM Data Source 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   ,$
```

Using a Generalized Record Type

How to:

Specify a Generalized Record Type

Example:

Using a Generalized Record Type

If your fixed-format sequential, VSAM, or ISAM data source has multiple record types that share the same layout, you can specify a single generalized segment that describes all record types that have 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 using a generalized segment, you identify RECTYPE values using the ACCEPT attribute. You can assign any value to the ALIAS attribute.

Syntax: How to Specify a Generalized Record Type

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

where:

`RECTYPE`

Is the required field name.

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.

`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.

`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 a Generalized Record Type

To illustrate the use of the generalized record type in a VSAM Master File, consider the following record layouts in the DOC data source. 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. A unique set of field names must be provided for record type M and record type I, although they have the same layout.

The generalized RECTYPE capability enables you to code just one set of field names that applies to the record layout for both record type M and 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 an ALIAS in a Report Request

Example:

Using a RECTYPE Value in a Display Command

Using a RECTYPE Value in a WHERE Test

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 a RECTYPE Value in a Display Command

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 a RECTYPE Value 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
```

Combining Multiply Occurring Fields and Multiple Record Types

In this section:

You can have two types of descendant segments in a single fixed-format sequential, VSAM, or ISAM data source:

- Descendant segments consisting of multiply occurring fields.
- Additional descendant segments consisting of multiple record types.

Describing a Multiply Occurring Field and 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 as descendant segments with an OCCURS attribute. The data source also contains two separate records, of types 02 and 03, which contain information that is related to that in record type 01.

The relationship between the records of various types is expressed as parent-child relationships. The children that contain record types 02 and 03 do 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

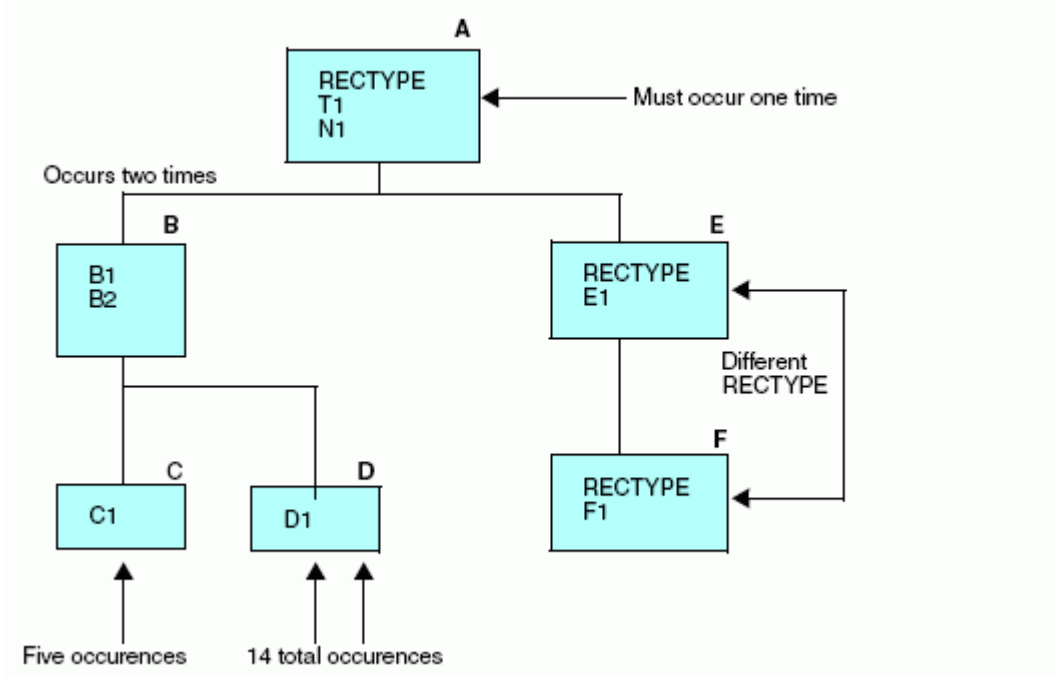
02	E1

03	F1

The description for this data source 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 that are related through their record types. The record type attribute consists of certain prescribed values, and is stored in a fixed location in the records. Records are expected to be retrieved in a given order. If the first record does not have a RECTYPE of 01, the record is considered to be a child without a parent. 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) or another 03.
- 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, 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.

Describing a VSAM Repeating Group With RECTYPEs

Example:

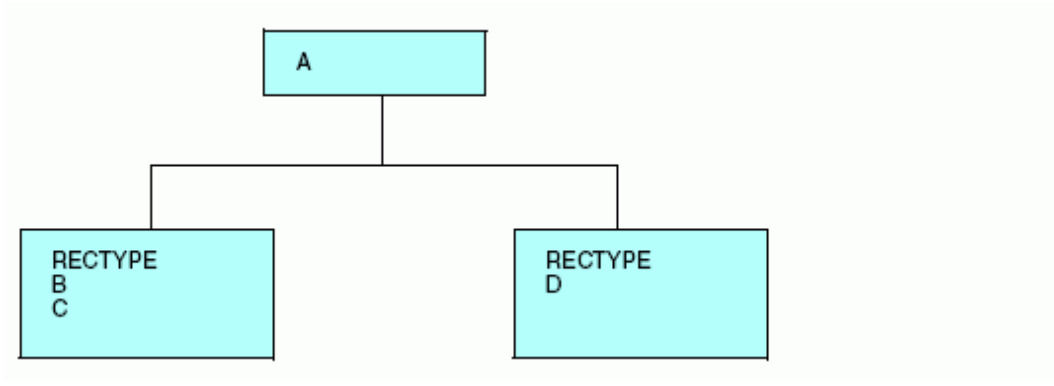
Describing a VSAM Repeating Group With RECTYPEs

Suppose you want to describe a data source that, schematically, looks like this:

```
A     RECTYPE  B C    RECTYPE   B C
A     RECTYPE  D      RECTYPE   D
```

You must 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.

The following diagram illustrates these segments.



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 each OCCURS segment depends on the RECTYPE indicator for its evaluation, the RECTYPE must appear at the start of the OCCURS segment. This enables you to describe complex data sources, including those with nested and parallel repeating groups that depend on RECTYPEs.

Example: Describing a VSAM Repeating Group With RECTYPEs

In this 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

```
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 a Repeating Group Using MAPFIELD

How to:

Describe a Repeating Group With MAPFIELD

Use MAPFIELD for a Descendant Repeating Segment in a Repeating Group

Example:

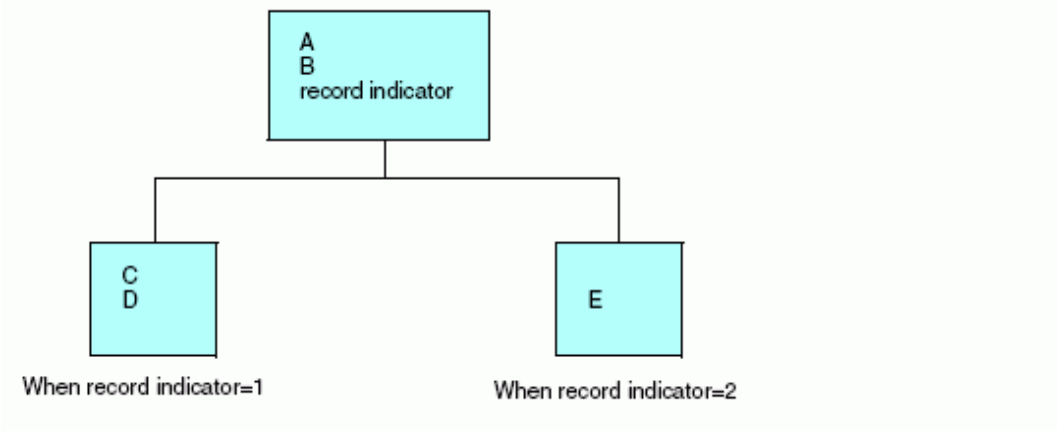
Using MAPFIELD and MAPVALUE

In another combination of record indicator and OCCURS, a record contains a record indicator that is followed by a repeating group. In this case, the record indicator is in the fixed portion of the record, not in each occurrence. Schematically, the record appears like this:

```
A B      record indicator (1)    C D    C D     C D
A B      record indicator (2)    E E
```

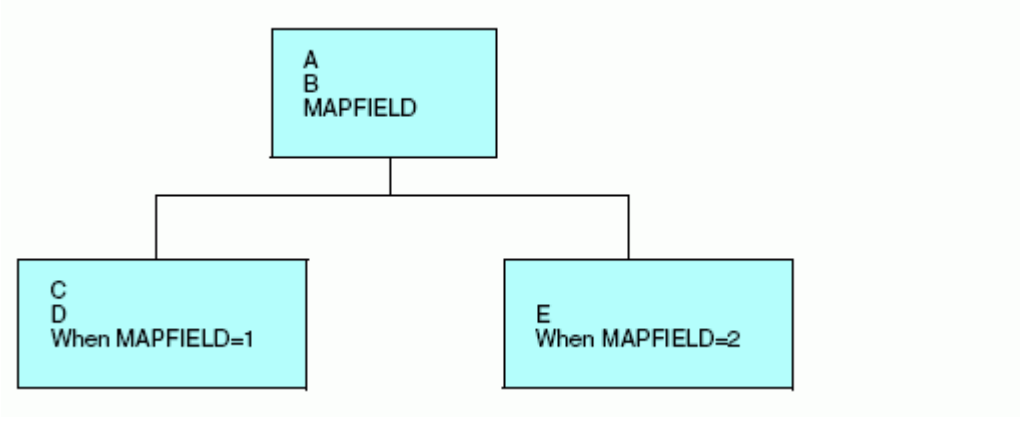
The first record contains header information, values for A and B, followed by an OCCURS segment of C and D that was identified by its preceding record indicator. The second record has a different record indicator and contains a different repeating group, this time for E.

The following diagram illustrates this relationship:



Since the OCCURS segments are identified by the record indicator rather than the parent A/B segment, you must use the keyword MAPFIELD. MAPFIELD identifies a field in the same way as RECTYPE, but since each OCCURS segment has its own value 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 is the record indicator; it may have any name.

Syntax: How to Describe a Repeating Group With MAPFIELD

```
FIELD = name, ALIAS = MAPFIELD, USAGE = format, ACTUAL = format,$
```

where:

name

Is the name you choose to provide for this field.

ALIAS

MAPFIELD is assigned as the alias of the field that is 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 MAPVALUE.

Syntax: How to Use MAPFIELD for a Descendant Repeating Segment in a Repeating Group

```
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 value, if an ACCEPT list is not specified. If there is an ACCEPT list, this can be any value.

`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.

Example: Using MAPFIELD and MAPVALUE

Using the sample data source at the beginning of this section, the Master File for this data source 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*.

MAPFIELD and MAPVALUE may be used with SUFFIX=FIX and SUFFIX=VSAM data sources.

Establishing VSAM Data and Index Buffers

How to:

Establish VSAM Data and Index Buffers

Two SET commands make it possible to establish DATA and INDEX buffers for processing VSAM data sources online.

The AMP subparameters BUFND and BUFNI enable z/OS BATCH users to enhance the I/O efficiency of TABLE, TABLEF, MODIFY, and JOIN against VSAM data sources by holding frequently used VSAM Control Intervals in memory, rather than on physical DASD. By reducing the number of physical Input/Output operations, you may improve job throughput. The SET commands enable users (in CMS, z/OS, 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.

Syntax: How to Establish VSAM Data and Index Buffers

```
{MVS|CMS} VSAM SET BUFND {n|8}
{MVS|CMS} VSAM SET BUFNI {n|1}
```

where:

n

Is the number of data or index buffers. BUFND=8 and BUFNI=1 (eight data buffers and one index buffer) are the default values.

Note: The AMODE setting controls whether the buffers are created above or below the line. SET AMODE=31 places the buffers above the line. SET AMODE=24 places the buffers below the line.

To determine how many buffers are in effect at any time, issue the query:

```
{MVS|CMS} VSAM SET ?
```

Using a VSAM Alternate Index

Example:

Describing a VSAM Alternate Index

Using IDCAMS

VSAM key-sequenced data sources support the use of alternate key indexes (keys). A key-sequenced VSAM data source consists of two components: 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 data source. Together, these components are referred to as the base cluster.

An alternate index is a separate, additional index structure that enables you to access records in a KSDS VSAM data source based on a key other than the data source's primary key. For instance, you may usually use a personnel data source sequenced by Social Security number, but occasionally need to retrieve records 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 data source.

The alternate index is a VSAM structure and is, consequently, created and maintained in the VSAM environment. It can, however, be described in your Master File, so that you can take advantage of the benefits of an alternate index.

The primary benefit of these indexes is improved efficiency. You can use it as an alternate, more efficient, retrieval sequence or 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 data sources with the JOIN command through this alternate index.

It is not necessary to identify the indexed view explicitly in order to take advantage of the alternate index. An alternate index is automatically used when described in the Master File.

To take advantage of a specific alternate index during a TABLE request, provide a WHERE or IF 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 see in the Master File in *Describing a VSAM Alternate Index* on page 236, the LNAME field is defined as an alternate index field. The records in the data source are retrieved according to their last names, and certain IF screens on the field LNAME result in direct reads. Note that if the alternate index field name is omitted, the primary key (if there is any) is used for a sequential or a direct read, and the alternate indexes are treated as regular fields.

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 ALIAS=KEY must be used to describe the primary key.

Only one record type can be referred to in the request when using alternate indexes, but there is no restriction on the number of OCCURS segments.

Note that the path name in the allocation is 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.)

Example: Describing a VSAM Alternate Index

Consider the following:

```
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 indexes. 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 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.

Example: Using IDCAMS

The following example demonstrates how to use IDCAMS 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.

Describing a Token-Delimited Data Source

How to:

Define a File With Delimiters

Example:

Defining a Delimiter

Separating Field Values for Missing Data

Reference:

Usage Notes for a Token-Delimited File

You can read files in which fields are separated by any type of delimiter including commas, tabs and other characters. Defining a Master File with the SUFFIX=DFIX attribute lets you specify any combination of characters as the field delimiter. Delimiters may consist of printable or non-printable characters, or any combination of printable and non-printable characters.

Syntax: How to Define a File With Delimiters

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

```
SUFFIX=DFIX
```

Describe the delimiter characters in a special field or group named DELIMITER. The delimiter characters are specified in the ALIAS attribute of this special field or group.

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

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

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

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

where:

`DELIMITER`

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

`delimiter`

Identifies a delimiter. For one or more printable characters, the value consists of the actual characters. The delimiter must be enclosed in single quotation marks if it includes characters used as delimiters in Master File syntax. For a non-printable character, the value is the decimal equivalent of the EBCDIC or ASCII representation of the character, depending on your operating environment.

`ufmt, afmt`

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

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

Reference: Usage Notes for a Token-Delimited File

- ❏ If the delimiter is alphanumeric and the delimiter value contains special characters (those used as delimiters in Master File syntax), it must be enclosed in single quotation marks.
- ❏ If the data is numeric and has a zoned format (ACTUAL=Z*n*), the data must be unsigned (can not contain a positive or negative value).
- ❏ Numeric (decimal) values may be used to represent any character, but are predominantly used for non-printable characters such as Tab. The numeric values may differ between EBCDIC and ASCII platforms.
- ❏ A delimiter is needed to separate field values. A pair of delimiters denotes a missing or default field value.
- ❏ Trailing delimiters are not necessary except that all fields must be terminated with the delimiter if the file resides in CMS or has fixed length records in z/OS.
- ❏ Only one delimiter field/group is permitted per Master File.
- ❏ Token-delimited files cannot be used in joins. They do not support the RECTYPE, POSITION, or OCCURS attributes.

Example: Defining a Delimiter

The following example shows a one-character alphanumeric delimiter:

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

The following example shows a two-character alphanumeric delimiter:

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

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

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

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

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

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

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

Example: Separating Field Values for Missing Data

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

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

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

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

The output is:

```
COUNTRY            CAR                NUMBER
GERMANY            VOLKSWAGEN          1111
GERMANY            BMW                    0
USA                CADILLAC           22222
USA                FORD                   0
USA                .                  44444
JAPAN              .                      0
ENGLAND                                   0
FRANCE             .                      0
```

Reading a Complex Data Source With a User-Written Procedure

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 a Non-FOCUS Data Source*.

- 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 Report Writer.

  The records, which can come from any source, are treated exactly as if they come from a FOCUS data source. The user routine must be coded as a subroutine in FORTRAN, COBOL, BAL, or PL/I, and passes data to the Report Writer 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
  ```

  The program named MYREAD is loaded and called to obtain data whenever there is a TABLE, TABLEF, MATCH, or GRAPH command for data source ABC.
- A decompression exit is also available for compressed VSAM data sources and flat data sources. This is called ZCOMP1. It uses SUFFIX=PRIVATE.

6 Describing a FOCUS Data Source

The following covers data description topics unique to FOCUS data sources:

- **Design tips.** Provides suggestions for designing a new FOCUS data source or changing the design of an existing data source.
- **Describing segments.** Contains information about Master File segment declarations for FOCUS data sources, including defining segment relationships, keys, and sort order using the SEGTYPE attribute, and storing segments in different locations using the LOCATION attribute.
- **Describing fields.** Contains information about Master File field declarations for FOCUS data sources, including the FIND option of the ACCEPT attribute, indexing fields using the INDEX attribute, redefining sequences of fields using the GROUP attribute, and the internal storage requirements of each data type defined by the FORMAT attribute, and of null values described by the MISSING attribute.
- **Describing two-gigabyte and partitioned data sources.** Contains information about Master File and Access File declarations for intelligently partitioned FOCUS data sources.

Topics:

Types of FOCUS Data Sources

In this section:

Using a SUFFIX=FOC Data Source

Using an XFOCUS Data Source

How to:

Enable Two-Gigabyte Support

Reference:

Usage Notes for a Two-Gigabyte FOCUS Data Source

The type of FOCUS data source you create depends on the amount of storage you require:

- FOCUS data sources with SUFFIX = FOC consist of 4K database pages.
- XFOCUS data sources with SUFFIX = XFOCUS consist of 16K database pages.

Using a SUFFIX=FOC Data Source

The FOCUS data source size can be a maximum of two gigabytes per physical data file. Through partitioning, one logical FOCUS data source can consist of up to 250 physical files of up to two gigabytes each, for a maximum of 500 gigabytes of real storage per logical data source.

Note that this discussion applies to any FOCUS data source that has extended beyond one gigabyte, but has not reached the two-gigabyte limit.

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

Syntax: How to Enable Two-Gigabyte Support

Issue the following command in the FOCPARM profile or, if the data source is running on a FOCUS Database Server, in HLIPROF

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

where:

`ON`

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

`OFF`

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

Reference: Usage Notes for a Two-Gigabyte FOCUS Data Source

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

  To add to a data source in z/OS, you must issue the LOAD command with the following syntax:

  ```
  LOAD NOCREATE
  ```

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

Using an XFOCUS Data Source

How to:

Specify an XFOCUS Data Source

Control the Number of Pages Used for XFOCUS Data Source Buffers

Create an XFOCUS Data Source

Reference:

Usage Notes for the XFOCUS Data Source

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

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

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

All existing commands that act on FOCUS files work on XFOCUS files. No new syntax is required, except for MDI options.

You can convert to an XFOCUS data source from a FOCUS data source by changing the SUFFIX in the Master File, and applying the REBUILD utility.

Syntax: How to Specify an XFOCUS Data Source

```
FILE = filename, SUFFIX = XFOCUS, $
```

where:

filename

Can be any valid file name.

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

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

```
SET XFOCUSBINS = n
```

where:

n

Is the number of pages used for XFOCUS data source buffers. Valid values are 16 to 1023. 64 is the default value.

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

Procedure: How to Create an XFOCUS Data Source

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

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

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

   ```
   CREATE FILE name
   ```

   where:

   name

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

Reference: Usage Notes for the XFOCUS Data Source

- On mainframe platforms, the LRECL and BLKSIZE are both 16384 (16K).
- The file type on VM is FOCUS and the extension on UNIX and Windows is foc.
- Alphanumeric fields with the format A4096 are supported. They are not limited to A3968 as in SUFFIX=FOC.
- The CALCFILE utility automatically adjusts the calculation algorithm to the suffix type and can be used to size the file.
- The USE command supports 250 files of mixed XFOCUS and FOC suffixes as long as each type of data source has its own Master File with the correct suffix (FOC for the FOCUS data sources, XFOCUS for the XFOCUS data sources).

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

Designing a FOCUS Data Source

In this section:

The 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 data source and tell you how you can change the design after the data source has been created.

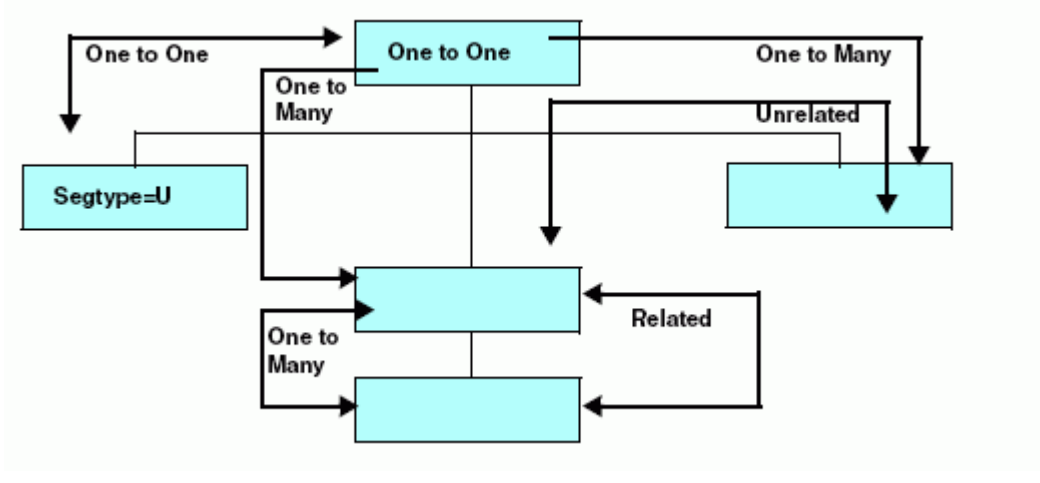
Data Relationships

The primary consideration when designing a data source is the set of relationships among the various fields. Before you create the Master File, 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 source, can that data source be joined to this one?

In general, 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 source should, in most cases, be retrieved in this way, and not redundantly maintained in two different data sources.
- 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. The name or 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

A FOCUS data source reads the root segment first, then traverses the hierarchy to satisfy your query. The smaller you make the root segment, the more root segment instances can be read at one time, and the faster records can be selected to process a query.

You can also improve record substitution efficiency by setting AUTOPATH. AUTOPATH is the automation of TABLE FILE *ddname.fieldname* syntax, where the 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 data source: you must 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 segment location of fields used in WHERE or IF tests is significant to the processing efficiency of a request. When a field fails a record selection test, there is no additional processing to that segment instance or its descendants. The higher the selection fields are in a data structure, the fewer the number of segments that must be read to determine a record's status.

After you have designed and created a data source, if you want to select records based on fields that are low in the data structure, you can rotate the data structure to place those fields temporarily higher by using an alternate view. Alternate views are discussed in Chapter 3, *Describing a Group of Fields*. For details on using alternate views in report requests, see the *Creating Reports* manual.

Use the following guidelines to help you design an efficient data structure:

- Limit the information in the root segment to what is necessary to identify the record and what is used often in screening conditions.
- Avoid unnecessary key fields. Segments with a SEGTYPE of S1 are 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 data source is SEGTYPE S1, for increased efficiency in MODIFY procedures that read transactions from unsorted data sources (FIXFORM).
- Use segments with a SEGTYPE of SH1 when adding and maintaining data in date sequence. In this case, a SEGTYPE of SH1 logically positions the most recent dates at the beginning of the data source, 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 the fields on which you perform frequent searches of unique instances. When you specify that a field be indexed, you construct and maintain a table of data values and their corresponding physical locations in the data source. Thus, indexing a field speeds retrieval.

  You can index any field you want, although it is advisable to limit the number of indexes in a data source since each index requires additional storage space. You must weigh the increase in speed against the increase in space.

Changing a FOCUS Data Source

After you have designed and created a FOCUS data source, 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 data source has been created.

Some characteristics whose attributes cannot be edited can be changed if you rebuild the data source using the REBUILD facility, as described in the *Maintaining Databases* manual. You can also use REBUILD to add new fields to a data source.

Describing a Single Segment

In this section:

In a segment description, you can describe key fields, sort order, and segment relationships. The number of segments cannot exceed 64 in a FOCUS data source. Non-FOCUS data sources can have up to 256 segments, and FOCUS data sources can participate in join or COMBINE structures that consist of up to 256 segments.

Using an indexed view reduces the maximum number of segments plus indexes to 191 for the structure being used. If AUTOINDEX is ON, you may be using an indexed view without specifically asking for one.

You can code LOCATION segments in a Master File to expand the file size by pointing to another physical file location.

You can also create a field to timestamp changes to a segment using AUTODATE.

Three additional segment attributes that describe joins between FOCUS segments, CRFILE, CRKEY, and CRSEGNAME, are described in Chapter 7, *Defining a Join in a Master File*.

Describing Keys, Sort Order, and Segment Relationships: SEGTYPE

How to:

Describe a Segment

Reference:

Usage Notes for SEGTYPE

FOCUS data sources 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 data sources to indicate a logical key sequence for that data source. SEGTYPE is discussed in Chapter 3, *Describing a Group of Fields*.

Syntax: How to Describe a Segment

The syntax of the SEGTYPE attribute when used for a FOCUS data source is

```
SEGTYPE = segtype
```

Valid values are:

`SH[n]`

Indicates that the segment's instances are sorted from highest to 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 lowest value to highest, 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 field and is therefore not sorted. New instances are added to the end of the segment chain. Any search starts at the current position.

S0 segments are often used to store text for applications where the text needs to be retrieved in the order entered, and the application does not need to search for particular instances.

`b/ (blank)`

Indicates that the segment has no key field, and is therefore not sorted. New instances are added to the end of the segment chain. Any search starts at the beginning of the segment chain.

SEGTYPE = b/ 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 root segment cannot be a b/ segment.

`U`

Indicates that the segment is unique, with 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 data source 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 7, *Defining a Join in a Master File*. The parent-child pointer is stored in the data source.

`KU`

Indicates that this is a cross-referenced segment joined to the data source 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 7, *Defining a Join in a Master File*. The parent-child pointer is stored in the data source.

`DKM`

Indicates that this is a cross-referenced segment joined to the data source 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 7, *Defining a Join in a Master File*. The parent-child pointer is resolved at run time, and therefore new instances can be added without rebuilding.

`DKU`

Indicates that this is a cross-referenced segment joined to the data source 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 7, *Defining a Join in a Master File*. The parent-child pointer is resolved at run time, and therefore new instances can be added without rebuilding.

`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 data source, 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 data source, 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 data source:

- **Alias.** SEGTYPE does not have an alias.
- **Changes.** You can change a SEGTYPE of S[*n*] or SH[*n*] to S0 or ø , or increase the number of key fields. To make any other change to SEGTYPE, you must use the REBUILD facility. REBUILD is described in the *Maintaining Databases* manual.

Describing a Key Field

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 = S*n*), and up to 99 keys in a segment sorted from high to low (SEGTYPE = SH*n*). 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 = ø (blank)).

Note: Text fields cannot be used as key fields.

Describing Sort Order

For segments that have key fields, 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 S*n* (where *n* is the number of keys), the instances are sorted using the concatenated values of the first *n* fields, beginning with the lowest value and continuing to the highest.
- **High to low.** By specifying a SEGTYPE of SH*n* (where *n* is the number of keys), the instances are sorted using the concatenated values of the first *n* fields, beginning with the highest value and continuing to the lowest.

  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 are the first ones encountered in a segment chain.

Understanding Sort Order

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 are 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 are 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 segments are 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 SEGTYPE is set to KLU in the join description.
- Physical one-to-many relationships are specified by setting SEGTYPE to any valid value beginning with S (such as S0, SH*n*, and S*n*) 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 7, *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 7, *Defining a Join in a Master File*.

Storing a Segment in a Different Location: LOCATION

How to:

Store a Segment in a Different Location

Example:

Specifying Location for a Segment

By default, all of the segments in a FOCUS data source are stored in one physical file. For example, all of the EMPLOYEE data source's segments are stored in the data source named EMPLOYEE.

Use the LOCATION attribute to specify that one or more segments be stored in a physical file separate from the main data source file. The LOCATION file is also known as a horizontal partition. You can use a total of 64 LOCATION files per Master File (one LOCATION attribute per segment, except for the root). This is helpful if you want to create a data source larger than the FOCUS limit for a single data source file, or if you want to store parts of the data source in separate locations for security or other reasons.

There are at least two cases in which to use the LOCATION attribute:

- Each physical file is individually subject to a maximum file size. You can use the LOCATION attribute to increase the size of your data source by splitting it into several physical files, each one subject to the maximum size limit. (Read Chapter 7, *Defining a Join in a Master File*, to learn if it is more efficient to structure your data as several joined data sources.)
- 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 data sources can be kept on separate storage media to save space or implement separate security mechanisms. In some situations, separating the segments into different data sources allows you to use different disk drives.

Divided data sources require more careful file maintenance. Be especially careful about procedures that are done separately to separate data sources, such as backups. For example, if you do backups on Tuesday and Thursday for two related data sources, and you restore the FOCUS structure using the Tuesday backup for one half and the Thursday backup for the other, there is no way of detecting this discrepancy.

Syntax: How to Store a Segment in a Different Location

```
LOCATION = filename [,DATASET = physical_filename]
```

where:

filename

Is the ddname of the file in which the segment is to be stored.

physical_filename

Is the physical name of the data source, dependent on the platform.

For example:

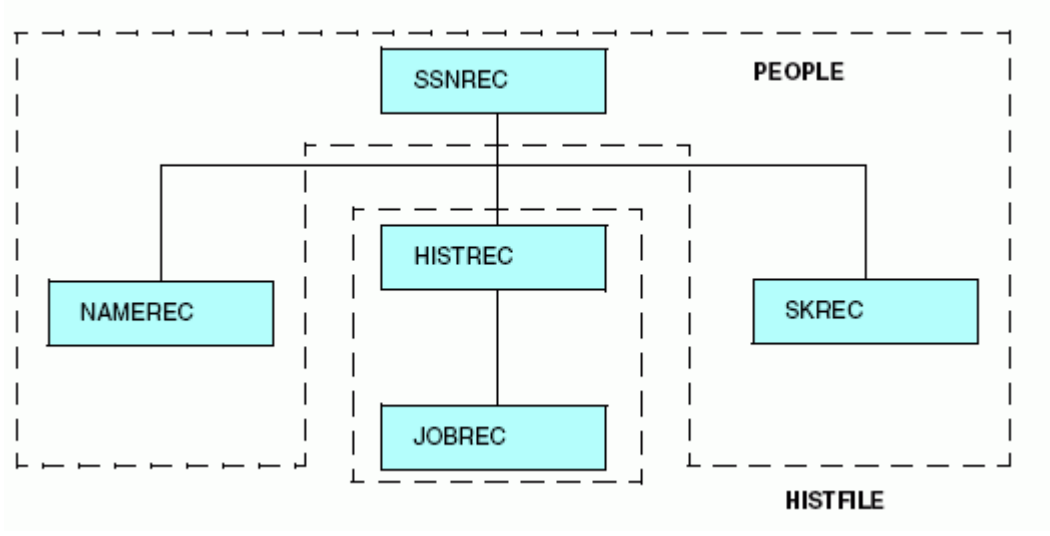
```
SEGNAME = HISTREC, SEGTYPE = S1, PARENT = SSNREC, LOCATION = HISTFILE, $
```

Example: Specifying Location for a Segment

The following illustrates the use of the LOCATION attribute:

```
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 data source. If no LOCATION attribute is specified for a segment, it is placed by default in the same file as its parent. In this example, you can assign the SKREC segment to a different file by specifying the LOCATION attribute in its declaration. However, it is recommended that you specify the LOCATION attribute, and not allow it to default.

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. Specify that you want a text field stored in a separate file by using the LOCATION attribute in the field definition.

For example, the text for DESCRIPTION is stored in a separate physical file named CRSEDESC:

```
FIELD = DESCRIPTION, ALIAS = CDESC, USAGE = TX50, LOCATION = 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.

In the following example, the text fields DESCRIPTION and TOPICS are stored in the LOCATION file CRSEDESC. The text field PREREQUISITE is stored in another file, PREREQS.

```
FIELD = DESCRIPTION , ALIAS = CDESC, USAGE = TX50, LOCATION = CRSEDESC,$
FIELD = PREREQUISITE, ALIAS = PREEQ, USAGE = TX50, LOCATION = PREREQS ,$
FIELD = TOPICS,       ALIAS =      , USAGE = TX50, LOCATION = CRSEDESC,$
```

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; you can use it 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 a FOCUS Master File are limited to 12 characters. Field names for text fields in an XFOCUS Master File are not subject to this 12 character limitation. However, for both types of data sources, alias names for these fields can be up to 66 characters.

Limits on the Number of Segments, LOCATION Files, Indexes, and Text Fields

Reference:

FDT Entries for a FOCUS or XFOCUS Data Source

The maximum number of segments in a Master File is 64. 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 and XFOCUS data sources. The FDT contains the names of the segments in the data source, the names of indexed fields, and the names of LOCATION files for text fields.

Reference: FDT Entries for a FOCUS or XFOCUS Data Source

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.

Determine the maximum number of LOCATION files for a data source using the following formula:

```
Available FDT entries = 189 - (Number of Segments + Number of 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).

Number of Segments

Is the number of segments in the Master File.

Number of Indexes

Is the number of indexed fields.

For example, a ten-segment data source with 2 indexed fields enables you to specify up to 52 LOCATION segments and/or LOCATION files for text fields (189 - (10 + 2)). Using the formula, the result equals 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 is included in the text location file count.

Specifying a Physical File Name for a Segment: DATASET

How to:

Use the DATASET Attribute on the Segment Level

Example:

Allocating a Segment Using the DATASET Attribute

In addition to specifying a DATASET attribute at the file level in a FOCUS Master File, you can specify the attribute on the segment level to specify the physical file name for a LOCATION segment, or a cross-referenced segment with field redefinitions.

For information on specifying the DATASET attribute at the file level, see Chapter 2, *Identifying a Data Source*.

Note:

- If you issue a USE command or explicit allocation for the file, a warning is issued that the DATASET attribute will be ignored.
- You cannot use both the ACCESSFILE attribute and the DATASET attribute in the same Master File.
- The MODIFY FIND function is not supported with the DATASET attribute. To use FIND with a data source, you must allocate the data source manually.

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

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

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

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

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

Note: If a DATASET allocation is in effect, you must issue a CHECK FILE command in order to override it by an explicit allocation command. The CHECK FILE command deallocates the allocation created by DATASET.

Syntax: How to Use the DATASET Attribute on the Segment Level

For a LOCATION segment:

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

For a cross-referenced segment:

```
SEGNAME=segname, SEGTYPE=segtype, PARENT=parent, [CRSEGNAME=crsegname,]
[CRKEY=crkey,] CRFILE=crfile, DATASET='filename1 [ON sinkname]',
  FIELD=...
```

where:

filename

Is the logical name of the LOCATION file.

physical_filename

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

sinkname

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

On z/OS, the syntax is

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

or

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

On CMS, the syntax is

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

Example: Allocating a Segment Using the DATASET Attribute

On z/OS:

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

On z/OS with SU:

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

On CMS:

```
FILE = ...
 SEGNAME=BDSEG,SEGTYPE=KU,CRSEGNAME=IDSEG,CRKEY=PRODMGR,
  CRFILE=PERSFILE,DATASET='IDSEG FOCUS A',
   FIELD=NAME,ALIAS=FNAME,FORMAT=A12,INDEX=I, $
```

Timestamping a FOCUS Segment: AUTODATE

How to:

Define an AUTODATE Field for a Segment

Example:

Defining an AUTODATE Field

Reference:

Usage Notes for AUTODATE

Each segment of a FOCUS data source can have a timestamp field that records the date and time of the last change to the segment. This field can have any name, but its USAGE format must be AUTODATE. The field is populated each time its segment instance is updated. The timestamp is stored as format HYYMDS, and can be manipulated for reporting purposes using any of the date-time functions.

In each segment of a FOCUS data source, you can define a field with USAGE = AUTODATE. The AUTODATE field cannot be part of a key field for the segment. Therefore, if the SEGTYPE is S2, the AUTODATE field cannot be the first or second field defined in the segment.

The AUTODATE format specification is supported only for a real field in the Master File, not in a DEFINE or COMPUTE command or a DEFINE in the Master File. However, you can use a DEFINE or COMPUTE command to manipulate or reformat the value stored in the AUTODATE field.

After adding an AUTODATE field to a segment, you must REBUILD the data source. REBUILD does not timestamp the field. It does not have a value until a segment instance is inserted or updated.

If a user-written procedure updates the AUTODATE field, the user-specified value is overwritten when the segment instance is written to the data source. No message is generated to inform the user that the value was overwritten.

The AUTODATE field can be indexed. However, it is recommended that you make sure the index is necessary, because of the overhead needed to keep the index up to date each time a segment instance changes.

If you create a HOLD file that contains the AUTODATE field, it is propagated to the HOLD file as a date-time field with the format HYYMDS.

Syntax: How to Define an AUTODATE Field for a Segment

```
FIELDNAME = fieldname, ALIAS = alias, {USAGE|FORMAT} = AUTODATE ,$
```

where:

fieldname

Is any valid field name.

alias

Is any valid alias.

Example: Defining an AUTODATE Field

Create the EMPDATE data source by performing a REBUILD DUMP of the EMPLOYEE data source and a REBUILD LOAD into the EMPDATE data source. The Master File for EMPDATE is the same as the Master File for EMPLOYEE, with the FILENAME changed and the DATECHK field added:

```
FILENAME=EMPDATE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,     ALIAS=EID,  FORMAT=A9,        $
 FIELDNAME=DATECHK,    ALIAS=DATE, USAGE=AUTODATE,   $
 FIELDNAME=LAST_NAME, ALIAS=LN,    FORMAT=A15,       $
   .
   .
   .
```

To add the timestamp information to EMPDATE, run the following procedure:

```
SET TESTDATE = 20010715
TABLE FILE EMPLOYEE
PRINT EMP_ID CURR_SAL
ON TABLE HOLD
END

MODIFY FILE EMPDATE
FIXFORM FROM HOLD
MATCH EMP_ID
ON MATCH COMPUTE CURR_SAL = CURR_SAL + 10;
ON MATCH UPDATE CURR_SAL
ON NOMATCH REJECT
DATA ON HOLD
END
```

Then reference the AUTODATE field in a DEFINE or COMPUTE command, or display it using a display command. The following request computes the number of days difference between the date 7/31/2001 and the DATECHK field:

```
DEFINE FILE EMPLOYEE
DATE_NOW/HYYMD = DT(20010731);
DIFF_DAYS/D12.2 =  HDIFF(DATE_NOW, DATECHK, 'DAY', 'D12.2');
END
TABLE FILE EMPDATE
PRINT DATECHK DIFF_DAYS
WHERE LAST_NAME EQ 'BANNING'
END
```

The output is:

```
DATECHK                     DIFF_DAYS
-------                     ---------
2001/07/15 15:10:37         16.00
```

Reference: Usage Notes for AUTODATE

- PRINT * and PRINT.SEG.*fld* print the AUTODATE field.
- To display the AUTODATE field on a CRTFORM, Winform, or in FSCAN, you must explicitly reference the AUTODATE field name in the request. CRTFORM * does not display the field. CRTFORM always treats the AUTODATE field as a display-only (D.) field.
- MODIFY FIXFORM and FREEFORM requests capture the system date and time per transaction.
- The FOCUS Database Server updates the AUTODATE field per segment using the date and time on the Server.
- Maintain processes AUTODATE fields at COMMIT time.
- DBA is permitted on the AUTODATE field; however, when unrestricted fields in the segment are updated, the system updates the AUTODATE field.
- The AUTODATE field does not support the following attributes: MISSING, ACCEPT, and HELPMESSAGE.

GROUP Attribute

In this section:

A group provides a convenient alternate name for one or more contiguous fields. The group redefines a sequence of fields and does not require any storage of its own.

Group fields enable you to:

- Join to multiple fields in a cross-referenced data source. Redefine the separate fields as a group, index this group key, and join it to the group field.
- Automatically use indexes in MODIFY when the root segment has multiple indexed fields that comprise the key. In this case you define the group as the key. The SEGTYPE then becomes S1, and MODIFY automatically uses the index.
- Use an indexed view in a TABLE request when records are selected based on an equality test on each component of the group. TABLE only uses one index for this type of retrieval, so you can index the group and enhance retrieval efficiency.

Describing a Group Field With Formats

How to:

Example:

Reference:

The component fields can contain alphanumeric or numeric data. However, the group field should always have an alphanumeric format. The length of the group field must be the sum of the actual lengths of the component fields. For example, integer fields always have an actual length of four bytes, regardless of the USAGE format that determines how many characters appear on a report.

Syntax: How to Define a Group Field With Formats

```
GROUP=groupname, [ALIAS=groupalias,] USAGE=An, [,FIELDTYPE=I] $
 FIELDNAME=field1, ALIAS=alias1, USAGE=fmt1,$
 FIELDNAME=field2, ALIAS=alias2, USAGE=fmt2,$
 .
 .
 .
 FIELDNAME=fieldn, ALIAS=aliasn, USAGE=fmtn,$
```

where:

groupname

Is the name of the group.

groupalias

Is an optional alias name for the group.

An

Is the format for the group field. Its length is the sum of the internal lengths of the component fields:

- Fields of type I have an internal length of 4.
- Fields of type F have an internal length of 4.
- Fields of type P that have a USAGE format of P15 or P16 or smaller have an internal length of 8, and fields of type P that have a USAGE format of P17 or greater have an internal length of 16.
- Fields of type D have an internal length of 8.
- Fields of type A have an internal length equal to the number of characters (n) in their USAGE formats.

Describing the group field with a format other than A does not generate an error message; however, it is not recommended and may produce unpredictable results.

field1, ..., fieldn

Are the component fields in the group.

alias1, ..., aliasn

Are the alias names for the component fields in the group.

fmt1, ..., fmtn

Are the USAGE formats of the component fields in the group.

FIELDTYPE=I

Creates a separate index for the group.

Reference: Usage Notes for Group Fields in FOCUS Data Sources

- When at least one component of the group has a numeric or date format, the group field does not appear correctly. However, the value of the group field is correct, and the individual fields appear correctly. Use this type of group in a TABLE request for indexed retrieval on more than one component field or to join to more than one component field.
- You can add or remove a non-key group field definition in a Master File at any time without impact. If the group field is indexed and you MODIFY the data source without the group field definition in the Master File, you must rebuild the index when you add the group field back to the Master File.
- The MISSING attribute is not supported on a group field.
- To use a group field that contains a numeric component in screening criteria, separate the component values with slashes. However, if the value of one the group components contains a slash '/', the slash may not be used as a delimiter and no test can be issued against this value.

  Note that using slashes makes it easier to specify values when the component fields contain trailing blanks, because you do not have to account for those blanks.

  The only masks supported in screening criteria for group fields are those that accept any combination of characters for all group components after the first component. For example, if the FULL_NAME group consists of LAST_NAME and FIRST_NAME, the following mask is supported:

  ```
  WHERE FULL_NAME EQ '$R$$$$*'
  ```

- To use a group field that contains only alphanumeric components in screening criteria, separating the component values with a slash is optional.
- A group format supports date display options.
- In MODIFY, you can match on the group field when it is the first field in the segment (S1). A MATCH on a component field in the group without matching on the group generates the following:

  ```
  (FOC439) WARNING. A MATCH CONDITION HAS BEEN ASSUMED FOR:
  ```

- If the group has an index, and the group components are also indexes, you can MATCH on the group level with no need to match on the group components.
- Although MODIFY enables you to update group components even if they are key fields, this is not recommended. Instead, use SCAN or FSCAN to update key fields.

- If both a group and the field following it are part of a key, the number specified for the SEGTYPE attribute must include the group field plus its component fields, plus the field following the group. For example, if the group key has two components and is followed by another field that is also part of the key, the SEGTYPE must be S4:

```
GROUP = ...,                ,$
  FIELDNAME = FIELDG1, ... ,$
  FIELDNAME = FIELDG2, ... ,$
FIELDNAME = FIELD3, ...    ,$
```

Example: Displaying an Alphanumeric Group Field

In the following group field definition, the group length is 25, the sum of the lengths of the LAST_NAME and FIRST_NAME fields:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,        ALIAS=EID,     FORMAT=A9,        $
 GROUP=FULL_NAME,                 ,     FORMAT=A25,       $
  FIELDNAME=LAST_NAME,    ALIAS=LN,      FORMAT=A15,      $
  FIELDNAME=FIRST_NAME,   ALIAS=FN,      FORMAT=A10,      $
```

The WHERE test on the group field does not need slashes between the component values, because both component fields are alphanumeric:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID HIRE_DATE
BY FULL_NAME
WHERE FULL_NAME GT 'CROSSBARBARA'
END
```

The output is:

```
FULL_NAME                 EMP_ID     HIRE_DATE
---------                 ------     ---------
GREENSPAN      MARY       543729165   82/04/01
IRVING         JOAN       123764317   82/01/04
JONES          DIANE      117593129   82/05/01
MCCOY          JOHN       219984371   81/07/01
MCKNIGHT       ROGER      451123478   82/02/02
ROMANS         ANTHONY    126724188   82/07/01
SMITH          MARY       112847612   81/07/01
SMITH          RICHARD    119265415   82/01/04
STEVENS        ALFRED     071382660   80/06/02
```

Example: Screening on a Group Field With an Integer Component

In the following group field definition, the group length is 29, the sum of the lengths of the LAST_NAME, FIRST_NAME, and HIRE_DATE fields. Because HIRE_DATE is an integer field, its internal length is 4:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,        ALIAS=EID,      FORMAT=A9,      $
 GROUP=FULL_NAME,                 ,      FORMAT=A29        $
  FIELDNAME=LAST_NAME,    ALIAS=LN,       FORMAT=A15,     $
  FIELDNAME=FIRST_NAME,   ALIAS=FN,       FORMAT=A10,     $
  FIELDNAME=HIRE_DATE,    ALIAS=HDT,      FORMAT=I6YMD,   $
```

In the following request, the component field values must be separated by slashes in the WHERE test. The request does not display the group field, because the integer component would not appear correctly:

```
TABLE FILE EMPGROUP
PRINT EMP_ID LAST_NAME FIRST_NAME HIRE_DATE
BY FULL_NAME NOPRINT
WHERE FULL_NAME GT 'CROSS/BARBARA/811102'
END
```

The output is:

```
EMP_ID     LAST_NAME        FIRST_NAME  HIRE_DATE
------     ---------        ----------  ---------
543729165  GREENSPAN        MARY         82/04/01
123764317  IRVING           JOAN         82/01/04
117593129  JONES            DIANE        82/05/01
219984371  MCCOY            JOHN         81/07/01
451123478  MCKNIGHT         ROGER        82/02/02
126724188  ROMANS           ANTHONY      82/07/01
112847612  SMITH            MARY         81/07/01
119265415  SMITH            RICHARD      82/01/04
071382660  STEVENS          ALFRED       80/06/02
```

Describing a Group Field as a Set of Elements

How to:

Describe a GROUP Field as a Set of Elements

Reference:

Usage Notes for Group Elements

Example:

Declaring a GROUP With ELEMENTS

A GROUP declaration in a Master File describes several fields as a single entity. One use of a group is to describe group keys in a VSAM data source. Sometimes referring to several fields by one group name facilitates ease of reporting.

Traditionally, when describing a GROUP field, you had to take account of the fact that while the USAGE and ACTUAL format for the GROUP field are both alphanumeric, the length portion of the USAGE format for the group had to be calculated as the sum of the component lengths, where each integer or single precision field counted as 4 bytes, each double precision field as 8 bytes, and each packed field counted as either 8 or 16 bytes depending on its size.

To avoid the need to calculate these lengths, you can use the GROUP ELEMENTS option, which describes a group as a set of elements without USAGE and ACTUAL formats.

Syntax: How to Describe a GROUP Field as a Set of Elements

```
GROUP=group1, ALIAS=g1alias,ELEMENTS=n1,$
   FIELDNAME=field11, ALIAS=alias11, USAGE=ufmt11, ACTUAL=afmt11, $
   .
   .
   .
   FIELDNAME=field1h, ALIAS=alias1h, USAGE=ufmt1h, ACTUAL=afmt1h, $
GROUP=group2,ALIAS=g2alias,ELEMENTS=n2,$
   FIELDNAME=field21, ALIAS=alias21, USAGE=ufmt21, ACTUAL=afmt21, $
   .
   .
   .
   FIELDNAME=field2k, ALIAS=alias2k, USAGE=ufmt2k, ACTUAL=afmt2k, $
```

where:

group1, group2

Are valid names assigned to a group of fields. The rules for acceptable group names are the same as the rules for acceptable field names.

n1, n2

Are the number of elements (fields and/or groups) that compose the group. If a group is defined within another group, the subgroup (with all of its elements) counts as one element of the parent group.

field11, field2k

Are valid field names.

alias11, alias2k

Are valid alias names.

ufmt11, ufmt2k

Are USAGE formats for each field.

afmt11, afmt2k

Are ACTUAL formats for each field.

Reference: Usage Notes for Group Elements

- To use the ELEMENTS attribute, the GROUP field declaration should specify only a group name and number of elements.
  - If a group declaration specifies USAGE and ACTUAL *without* the ELEMENTS attribute, the USAGE and ACTUAL are accepted as specified, even if incorrect.
  - If a group declaration specifies USAGE and ACTUAL *with* the ELEMENTS attribute, the ELEMENTS attribute takes precedence.
- Each subgroup counts as one element. Its individual fields and subgroups do not count in the number of elements of the parent group.

Example: Declaring a GROUP With ELEMENTS

In the following Master File, GRP2 consists of two elements, fields FIELDA and FIELDB. GRP1 consists of two elements, GRP2 and field FIELDC. Field FIELDD is not part of a group:

```
FILENAME=XYZ      , SUFFIX=FIX      , $
  SEGMENT=XYZ, SEGTYPE=S2, $
GROUP=GRP1,ALIAS=CCR,ELEMENTS=2,$
   GROUP=GRP2,ALIAS=CC,ELEMENTS=2,$
    FIELDNAME=FIELDA, ALIAS=E01, USAGE=A10, ACTUAL=A10, $
    FIELDNAME=FIELDB, ALIAS=E02, USAGE=A16, ACTUAL=A16, $
    FIELDNAME=FIELDC, ALIAS=E03, USAGE=P27, ACTUAL=A07, $
    FIELDNAME=FIELDD, ALIAS=E04, USAGE=D7, ACTUAL=A07, $
```

The following chart shows the offsets and formats of these fields.

Field Number	Field Name	Offset	USAGE	ACTUAL
1	GRP1	0	A42 - Supports 16 characters for FIELDC (P27)	A33
2	GRP2	0	A26	A26
3	FIELDA	0	A10	A10
4	FIELDB	10	A16	A16
5	FIELDC	26	P27	A7
6	FIELDD	42	D7	A7

ACCEPT Attribute

How to:

Specify Data Validation

ACCEPT is an optional attribute that you can use to validate data that is entered into a field using a MODIFY procedure. For a description of its use with all types of data sources, see Chapter 4, *Describing an Individual Field*. However, ACCEPT has a special option, FIND, that you can use only with FOCUS data sources. FIND enables you to verify incoming data against values stored in another field.

Syntax: How to Specify Data Validation

```
ACCEPT = list
ACCEPT = range
ACCEPT = FIND (sourcefield [AS targetfield] IN file)
```

where:

list

Is a list of acceptable values. See Chapter 4, *Describing an Individual Field*.

range

Gives the range of acceptable values. See Chapter 4, *Describing an Individual Field*.

FIND

Verifies the incoming data against the values in an index in a FOCUS data source.

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.

targetfield

Is the name of the field that contains the acceptable data values. This field must be indexed.

file

Is the name of the file describing the data source that contains the indexed field of acceptable values.

INDEX Attribute

In this section:

Joins and the INDEX Attribute

FORMAT and MISSING: Internal Storage Requirements

How to:

Specify Field Indexing

Reference:

Usage Notes for INDEX

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 must create an index to:

- Join two segments based on equality. The cross–referenced field in a joined FOCUS data source must be indexed, as described in *Describing a Single Segment* on page 252 (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 Chapter 3, *Describing a Group 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
```

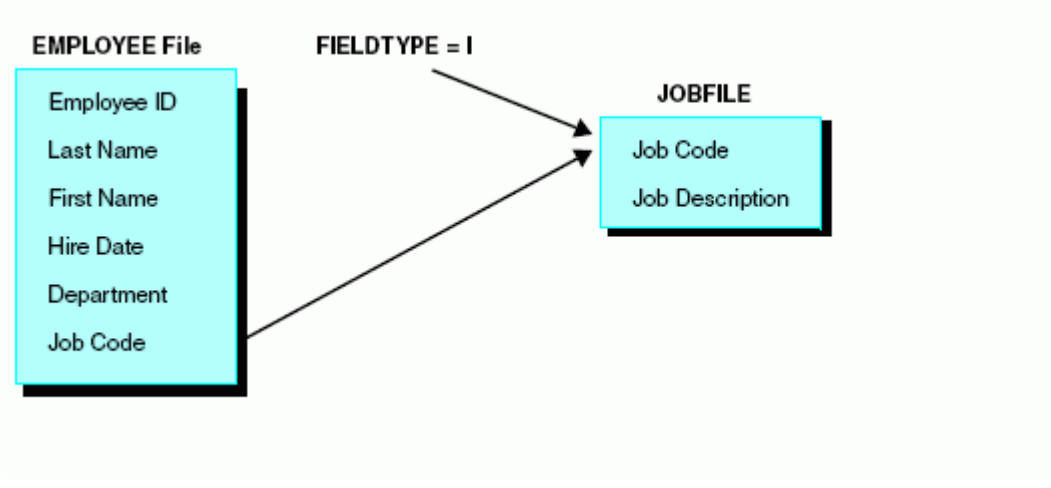
Text fields cannot be indexed. The maximum field name length for indexed fields in a FOCUS data source is 12 characters. The maximum field name length for indexed fields in an XFOCUS data source is 66 characters.

For example:

```
FIELDNAME = JOBCODE, ALIAS = CJC, FORMAT = A3, INDEX = I, $
```

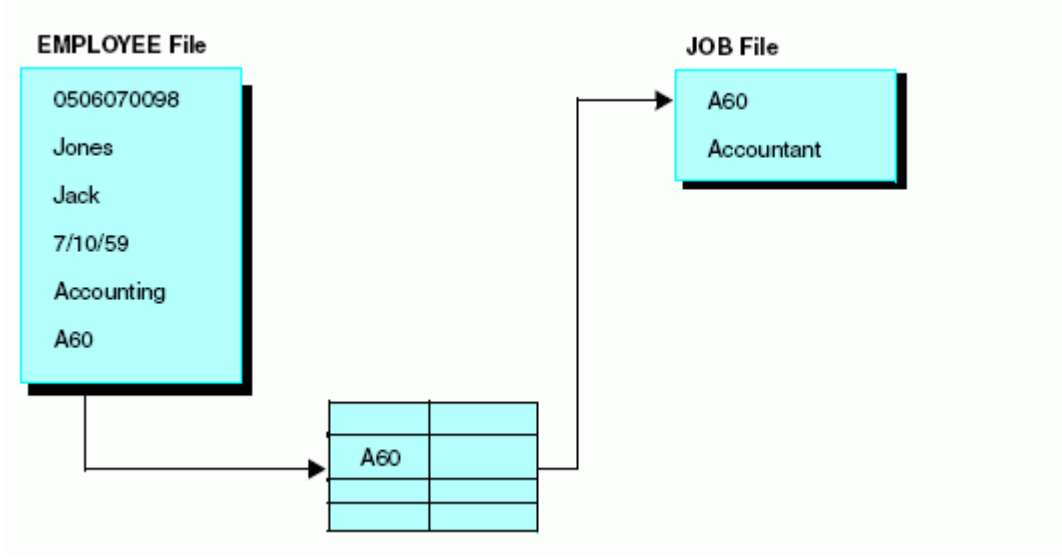
Joins and the INDEX Attribute

In order to be cross-reference a segment with a static cross-reference, a dynamic cross-reference, or an equijoin, at least one field in the segment must be indexed. This field, called the cross-referenced field, shares values with a field in the host data source. Only the cross-referenced segment requires an indexed field, shown as follows:



Other data sources 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 data source.

The value for the field named JOBCODE in the EMPLOYEE data source is matched to the field named JOBCODE in the JOBFILE data source by using the index for the JOBCODE field in the JOBFILE data source, as follows:



Indexes are stored and maintained as part of the FOCUS data source. 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 data sources which have data items in common with indexed fields in existing data sources can be added at any time.

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 the equivalent value of blank, the index is longer maintained. If you no longer need the index, use the REORG option of the REBUILD facility to recover space occupied by the index after you remove the INDEX attribute. REBUILD is described in the *Maintaining Databases* manual.

  To turn off indexing temporarily—for example, to load a large amount of data into the data source quickly—you can remove the INDEX attribute before loading the data, restore the attribute, and then use the REBUILD command with the INDEX option to create the index. This is known as post-indexing the data source.

  You can index the field after the data source has already been created and populated with records, by using the REBUILD facility with the INDEX option. A total of seven indexes may be added to the data source after the file is created using REBUILD INDEX. After seven indexes have been added to a data source in this way, you must use the REORG option of the REBUILD facility before adding an eighth; otherwise the following diagnostic message is issued:

  ```
  (FOC720) THE NUMBER OF INDEXES ADDED AFTER FILE CREATION EXCEEDS 7
  ```

- **Maximum number.** The total of indexes, text fields, and segments cannot exceed 189 (of which a maximum of 64 can be segments and text LOCATION files).

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 8 or 16 bytes and represent 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 (or JAN 1901, depending on the date format).

- Date-time fields are stored in 8 or 10 bytes depending on whether the time component specifies microseconds.
- Alphanumeric fields are stored as characters in the specified number of bytes.
- Variable length alphanumeric fields are stored as characters in the specified number of bytes plus two additional bytes to specify the length of the character string stored in the field.
- Missing values are represented internally by a dot (.) for alphanumeric fields, and as the value -9998998 for numeric fields.

Describing a Partitioned FOCUS Data Source

In this section:

FOCUS data sources can consist of up to 250 physical files. The horizontal partition is a slice of one or more segments of the entire data source structure. Note, however, that the number of physical files associated with one FOCUS data source is the sum of its partitions and LOCATION files. This sum must be less than or equal to 250. FOCUS data sources can grow in size over time, and can be repartitioned based on the requirements of the application.

Note: You do not have to partition your data source. If you choose not to, your application automatically supports FOCUS data sources larger than one gigabyte when you set the FOC2GIGDB parameter to ON.

Intelligent Partitioning

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

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

To take advantage of the partitioning feature, you must:

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

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

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

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

Specifying an Access File in a FOCUS Master File

How to:

Specify an Access File for a FOCUS Data Source

Reference:

Usage Notes for Partitioned FOCUS Data Sources

Example:

Master File for the VIDEOTR2 Partitioned Data Source

Using a Partitioned Data Source

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

In prior releases, the Access File for a FOCUS data source was space delimited and could describe the files associated with several Master Files. The name of the Access File was arbitrary and was specified with the ACCESS = attribute in the Master File.

The new syntax is comma-delimited, similar to the Master File syntax. It can describe the files needed for accessing only one Master File. The name of the Access File must be the same as the Master File name, and it must be specified with the ACCESS = attribute in the Master File.

Both types of syntax are supported in the FOCUS 7.6 track, but starting with FOCUS 7.7, only the comma-delimited syntax will be supported.

Both types of Access File syntax follow with examples relevant to each.

Syntax: How to Specify an Access File for a FOCUS Data Source

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

where:

filename

Is the file name of the partitioned data source.

accessfile

Is the name of the Access File. For a space-delimied Access File, this can be any valid name. For a comma-delimited Access File, this must be the same as the Master File name.

Reference: Usage Notes for Partitioned FOCUS Data Sources

- Concatenation of multiple partitions in one request is only valid for reporting. To MODIFY or REBUILD a partitioned data source, you must explicitly allocate and MODIFY, Maintain, or REBUILD one partition at a time.
- The order of precedence for allocating data sources is:
  - A USE command that is in effect has the highest precedence. It overrides an Access File or an explicit allocation for a data source.
  - An Access File overrides an explicit allocation for a data source.
- A DATASET attribute cannot be used in the same Master File as an ACCESSFILE attribute.
- Commands that alter a data source (for example, MODIFY, Maintain, and REBUILD) do not use the Access File. If you use a Master File that contains an ACCESSFILE attribute with a command that alters the data source, the following warning message appears:

  ```
  (FOC1968)ACCESS FILE INFORMATION IN MASTER %1 WILL NOT BE CONSIDERED
  ```

- The CREATE FILE command automatically issues a dynamic allocation for the data source it creates, and this allocation takes precedence over the ACCESSFILE attribute. In order to use the ACCESSFILE attribute after issuing a CREATE FILE command, you must first free this automatic allocation.
- When the type of command changes from reading a data source to writing to a data source, or vice versa (for example, Maintain to TABLE), the Master File is reparsed.
- When a cross-referenced Master File includes an ACCESSFILE attribute, the host Master File cannot rename the cross-referenced fields.

Example: Master File for the VIDEOTR2 Partitioned Data Source

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

Example: Using a Partitioned Data Source

The following illustrates how to use an intellgently partitioned data source. The Access File for the VIDEOTR2 data source describes three partitions based on DATE:

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

The output is:

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

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

The FOCUS Space-Delimited Access File

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

The Access File describes how to locate, concatenate, join, and select the appropriate physical data files for retrieval requests against one or more FOCUS data sources. Access Files are optional, except for retrieval requests using intelligent partitioning.

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

Use an Access File to take advantage of the following data source features:

- **Horizontal and vertical partitioning.** A data source can consist of several separate files, or partitions, each of which contains the data source records for a specific time period, region, or other element. It can also have LOCATION files for individual segments (horizontal partitions). The Access File describes how to concatenate the separate data sources.
- **Joins.** If joined data sources are partitioned, the Access File describes how to concatenate the separate data sources in the join.

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

Note: On z/OS, the Access File must be a member of a data set concatenated in the allocation for ddname ACCESS. On VM/ESA, the Access File must have the file type ACCESS. FOCSQL cannot be used as the file type. The Access File has the same DCB attributes as the Master File.

FOCUS Space-Delimited Access File Attributes

In this section:

Describing Joined Data Sources in a Space-Delimited Access File

How to:

Create a Space-Delimited Access File

Example:

Describing an Intelligent Partition in a FOCUS Space-Delimited Access File

Describing a Partition With a LOCATION File in a Space-Delimited Access File

The Access File can include the following attributes:

Attribute	Synonyms	Description
MASTERNAME	MASTER	Master File entry.
DATANAME	DATA	Name of the physical file.
WHERE		WHERE criteria.
LOCATION		Segment location.

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

Syntax: How to Create a Space-Delimited Access File

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

where:

`MASTERNAME`

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

`filename1, filename2`

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

`DATANAME`

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

`dataname1, dataname2`

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

test

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

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

where:

fieldname, *fieldname1*, *fieldname2* are field names in the Master File.

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

value1, *value2*, *value3*, *value4* are valid values for their corresponding fields.

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

LOCATION

Is the attribute that identifies a separately stored segment.

locationnamea, locationnameb

Are the values of the LOCATION attributes from the Master File. Segment locations must map one-to-one to horizontal partitions.

datanamea, datanameb

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

Example: Describing an Intelligent Partition in a FOCUS Space-Delimited Access File

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

For z/OS:

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

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

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

For CMS:

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

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

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

Example: Describing a Partition With a LOCATION File in a Space-Delimited Access File

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

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

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

For z/OS:

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

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

For CMS:

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

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

Describing Joined Data Sources in a Space-Delimited Access File

Example:

Joining Two Partitioned Data Sources

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

If one of the joined data sources is horizontally partitioned, only that data source needs an Access File to implement the join.

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

Example: Joining Two Partitioned Data Sources

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

For z/OS:

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

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

For CMS:

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

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

FOCUS Comma-Delimited Access File Attributes

How to:

Create a Comma-Delimited FOCUS Access File

Reference:

Access File Attributes for a Comma-Delimited Access File

Example:

Comma-Delimited Access File for the VIDEOTR2 Data Source

Every request supplies the name of a Master File. The Master File is read and the declarations in it are used to access the data source. If the Master File contains an ACCESS= attribute that references an Access File, the Access File is read and used to locate the correct data sources. With the comma-delimited syntax, the Access File must have the same name as the Master File. With the space-delimited syntax, the Access File can have any name. If there is no Access File with the same name as the ACCESS= attribute in the Master File, the request is processed with the Master File alone.

Reference: Access File Attributes for a Comma-Delimited Access File

Each comma-delimited Access File describes the files and MDIs for one Master File, and that Master File must have the same file name as the Access File.

All attribute/value pairs are separated by an equal sign (=), and each pair in a declaration is delimited with a comma (,). Each declaration is terminated with the comma dollar sign (,$).

1. Each Access File starts with a declaration that names its corresponding Master File.
2. Next comes the DATA declaration that describes the location of the physical file. If the file is partitioned, it has multiple DATA declarations.

   If the file is *intelligently* partitioned so that an expression describes which data values reside in each partition, the DATA declaration has a WHERE phrase that specifies this expression.
3. If the data source has LOCATION segments the LOCATION declaration names a location segment. Its corresponding LOCATIONDATA declaration points to the physical LOCATION file.
4. If the data source has an MDI, the Access File has an MDI declaration that names the MDI and its target segment, followed by declarations that name the dimensions of the MDI, followed by the MDIDATA declaration that points to the physical MDI file. If the MDI is partitioned, there are multiple MDIDATA declarations for the MDI.

Syntax: How to Create a Comma-Delimited FOCUS Access File

Master File declaration:

```
MASTER=mastername,$
```

where:

mastername

Indicates the name of the Master File with which this Access File is associated. It is the same value included in the Master File ACCESS=*filename* attribute, used to indicate both the existence of the Access File and its name.

```
DATA=file_specification,
   [WHERE= expression; ,]$
[DATA=file_specification,
   [WHERE= expression; ,]$ ...]
```

where:

file_specification

Points to the file location. This is a complete file specification. There are can be up to 250 DATA declarations (partitions) in a single Access File. With XFOCUS data sources, this supports creation of a 4 Terabtye database. Using FOCUS data sources, a 500 GB database can be constructed.

The WHERE clause is the basis of the Intelligent Partitioning feature. The expression is terminated with the semi-colon and the entire declaration with the comma/dollar sign. WHERE expressions of the following type are supported:

```
WHERE = field  operator value1 [ OR  value2...]; ,$
```

```
WHERE = field FROM value1 TO value2 [AND FROM value3 TO value4];,$
```

Expressions can be combined with the AND operator.

Location File declarations:

```
LOCATION=location_segment_name,$
  LOCATIONDATA=location_segment_file_specification,$
```

where:

location_segment_name

Is the name of the segment stored in the location file.

location_segment_file_specification

Is the full file specification for the physical file the segment is located in.

MDI declarations:

```
MDI=mdiname, TARGET_OF = segname,$
 DIM = [filename.]fieldname [, MAXVALUES = n] [, WITHIN = dimname],$
 [DIM = [filename.]fieldname [, MAXVALUES = n] [, WITHIN = dimname] ,$
    ...]
 MDIDATA=mdi_file_specification,$
 [MDIDATA=mdi_file_specification,$ ...]
```

where:

mdiname

Is the name of the MDI.

segname

Is the name of the target segment

filename

Is the name of the file where an MDI dimension resides.

fieldname

Is the name of a field that is a dimension of the MDI.

n

Is the number of distinct values in the dimension. When the MDI is created, the actual dimension value will be converted to an integer of length 1, 2, or 4 bytes, and this number will be stored in the index leaf.

mdi_file_specification

Is the fully-qualified specification of the physical MDI file. If the MDI is partitioned, it is the specification for one partition of the MDI. An MDI can have up to 250 MDIDATA declarations (partitions). An Access File can have an unlimited number of MDIs.

dimname

Defines a hierarchy of dimensions. This dimension is defined within the *dimname* dimension. For example, CITY WITHIN STATE.

Example: Comma-Delimited Access File for the VIDEOTR2 Data Source

VIDEOTR2 is an intelligently partitioned FOCUS data source. The Master File has an ACCESS=VIDEOTR2 attribute:

```
FILENAME=VIDEOTR2,  SUFFIX=FOC,    ACCESS=VIDEOTR2
SEGNAME=CUST,       SEGTYPE=S1
 FIELDNAME=CUSTID,        ALIAS=CIN,          FORMAT=A4,       $
 FIELDNAME=LASTNAME,      ALIAS=LN,           FORMAT=A15,      $
 FIELDNAME=FIRSTNAME,     ALIAS=FN,           FORMAT=A10,      $
 FIELDNAME=EXPDATE,       ALIAS=EXDAT,        FORMAT=YMD,      $
 FIELDNAME=PHONE,         ALIAS=TEL,          FORMAT=A10,      $
 FIELDNAME=STREET,        ALIAS=STR,          FORMAT=A20,      $
 FIELDNAME=CITY,          ALIAS=CITY,         FORMAT=A20,      $
 FIELDNAME=STATE,         ALIAS=PROV,         FORMAT=A4,       $
 FIELDNAME=ZIP,           ALIAS=POSTAL_CODE,  FORMAT=A9,       $
 FIELDNAME=EMAIL,         ALIAS=EMAIL,        FORMAT=A18,      $
SEGNAME=TRANSDAT, SEGTYPE=SH1,  PARENT=CUST
 FIELDNAME=TRANSDATE,    ALIAS=OUTDATE,   FORMAT=HYYMDI,
   MISSING=ON, $
SEGNAME=SALES,    SEGTYPE=S2,   PARENT=TRANSDAT
 FIELDNAME=TRANSCODE,    ALIAS=TCOD,    FORMAT=I3,           $
 FIELDNAME=QUANTITY,     ALIAS=NO,      FORMAT=I3S,          $
 FIELDNAME=TRANSTOT,     ALIAS=TTOT,    FORMAT=F7.2S,        $
SEGNAME=RENTALS,  SEGTYPE=S2,   PARENT=TRANSDAT
 FIELDNAME=MOVIECODE,    ALIAS=MCOD,      FORMAT=A6, INDEX=I, $
 FIELDNAME=COPY,         ALIAS=COPY,      FORMAT=I2,          $
 FIELDNAME=RETURNDATE,   ALIAS=INDATE,    FORMAT=YMD,         $
 FIELDNAME=FEE,          ALIAS=FEE,       FORMAT=F5.2S,       $
 DEFINE DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
```

The following shows the Access File, named VIDEOTR2, on z/OS:

```
MASTER=VIDEOTR2 ,$
  DATA=USER1.VIDPART1.FOCUS,
    WHERE=DATE EQ 1991;,$

  DATA=USER1.VIDPART2.FOCUS,
    WHERE=DATE FROM 1996 TO 1998; ,$

  DATA=USER1.VIDPART3.FOCUS,
   WHERE=DATE FROM 1999 TO 2000;,$
```

The following shows the Access File, named VIDEOTR2, on CMS:

```
MASTER=VIDEOTR2 ,$
  DATA='VIDPART1 FOCUS A',
    WHERE=DATE EQ 1991;,$

  DATA='VIDPART2 FOCUS A',
    WHERE=DATE FROM 1996 TO 1998; ,$

  DATA='VIDPART3 FOCUS A',
   WHERE=DATE FROM 1999 TO 2000;,$
```

Multi-Dimensional Index (MDI)

In this section:

A multi-dimensional index (MDI) enables you to efficiently and flexibly retrieve information you need for business analysis. It looks at data differently from transaction processing systems in which the goal is to retrieve records based on a key. (FOCUS uses a B-tree index for this type of retrieval). The MDI is for retrieval only. It is not used for MODIFY or Maintain requests.

Business analysts may be interested in specific facts (data values, also called measures) about multiple categories of data in the data source. Categories of data, such as region or department, are referred to as dimensions. A multi-dimensional index uses dimensions and all of their hierarchical relationships to point to specific facts.

The MDI is a multi-field index that contains at least two dimensions. This index behaves like a virtual cube of values that intersect at measures of interest. The more dimensions used in a query, the better the retrieval performance.

For example, suppose that the CENTORD data source has an MDI with dimensions STATE, REGION, and PRODCAT. The MDI is used to retrieve the facts (LINEPRICE and QUANTITY data) that lie at the intersection of the dimension values specified in the following request:

```
TABLE FILE CENTORD
SUM QUANTITY LINEPRICE
WHERE REGION EQ 'EAST'
WHERE STATE EQ 'DC'
WHERE PRODCAT EQ 'Cameras'
END
```

The MDI also provides the following other retrieval enhancing features: MDI JOIN, Dimensional JOIN, MDI WITHIN, MAXVALUES, MDI Encoding, and AUTOINDEX for MDI.

Specifying an MDI in the Access File

How to:

Specify an MDI in an Access File

Example:

Defining an MDI on UNIX

Defining an MDI on Windows

Defining an MDI on CMS

Defining an MDI on z/OS

All MDI attributes are specified in the Access File for the data source. The only attribute needed in the Master File is the ACCESSFILE attribute to point to the Access File containing the MDI specifications.

An MDI can be partitioned into multiple MDI files. However, even if the data source on which the MDI is built is partitioned, each MDI partition spans all data source partitions.

Syntax: How to Specify an MDI in an Access File

```
MASTER = masterfile,$
    DATA = database_filename1,$
     .
     .
     .
    DATA = database_filenamen,$
    MDI = mdiname,
        TARGET_OF = segname,$
        DIM = field1, [MAXVALUES=n1] [, WITHIN = dimname1],$
          .
          .
          .
        DIM = fieldn [MAXVALUES = nn] [, WITHIN = dimnamen],$

        MDIDATA = mdifile1,$
          .
          .
          .
        MDIDATA = mdifilen,$
```

where:

masterfile

Is the Master File name.

database_filename1, ..., database_filenamen

Are fully qualified physical file names in the syntax native to your operating environment. If the name contains blanks or special characters, it must be enclosed in single quotation marks. Multiple DATA declarations describe concatenated partitions.

mdiname

Is the logical name of the MDI.

segname

Is the segment that contains the facts pointed to by the MDI. If the target data is distributed among several segments, the target should be the top segment that contains MDI data in order to avoid the multiplicative effect.

field1, ..., fieldn

Are the fields to use as dimensions. At least two dimensions are required for an MDI.

mdifile1, ..., mdifilen

Are fully qualified physical file names for the MDI in the syntax native to your operating environment. If the name contains blanks or special characters, it must be enclosed in single quotation marks. Multiple MDIDATA declarations describe concatenated partitions.

n1, ..., nn

Is the number of distinct values the field can have. This number must be a positive integer.

dimname1, ..., dimnamen

Defines a hierarchy of dimensions. This dimension is defined within the *dimname* dimension. For example, CITY WITHIN STATE.

Example: Defining an MDI on UNIX

This example shows an MDI with two partitions:

```
MASTERNAME = CAR,$
 DATA = /user1/car.foc,$
 MDI = carmdi,
   TARGET_OF = ORIGIN,$
   DIM = CAR,$
   DIM = COUNTRY,$
   DIM = MODEL,$
   MDIDATA = /user1/car1.mdi,$
   MDIDATA = /user1/car2.mdi,$
```

Example: Defining an MDI on Windows

This example shows an MDI with two partitions:

```
MASTERNAME = CAR,$
 DATA = c:\user1\car.foc,$
 MDI = carmdi,
   TARGET_OF = ORIGIN,$
   DIM = CAR,$
   DIM = COUNTRY,$
   DIM = MODEL,$
   MDIDATA = c:\user1\car1.mdi,$
   MDIDATA = c:\user1\car2.mdi,$
```

Example: Defining an MDI on CMS

This example shows an MDI with two partitions:

```
MASTER = CAR,$
 DATA = 'CAR FOCUS M',$
 MDI = CARMDI,
   TARGET_OF = ORIGIN,$
   DIM = CAR,$
   DIM = COUNTRY,$
   DIM = MODEL,$
   MDIDATA = 'CARMDI1 MDI M',$
   MDIDATA = 'CARMDI2 MDI M',$
```

Example: Defining an MDI on z/OS

This example shows an MDI with two partitions:

```
MASTER = CAR,$
 DATA = USER1.CAR.FOCUS,$
 MDI = CARMDI,
   TARGET_OF = ORIGIN,$
   DIM = CAR,$
   DIM = COUNTRY,$
   DIM = MODEL,$
   MDIDATA = USER1.CAR1.MDI,$
   MDIDATA = USER1.CAR2.MDI,$
```

Creating a Multi-Dimensional Index

Each MDI is specified in an Access File for the data source. FOCUS uses the Access File in its retrieval analysis for each TABLE request.

You then use the REBUILD MDINDEX command to build the MDI. The MDI has the following DCB attributes: RECFM=F,LRECL=4096,BLKSIZE=4096.

A multi-dimensional index gives complex queries high-speed access to combinations of dimensions across data sources. If you know what information users want to retrieve and why, you can make intelligent choices about index dimensions.

An Access File can define more than one MDI. If the Access File defines multiple MDIs, the AUTOINDEX facility chooses the best index to use for each query.

The first step in designing an MDI is to find out what kind of information users need from the data source. You can get advice about your MDIs directly from FOCUS.

Choosing Dimensions for Your Index

Reference:

Guidelines for a Multi-Dimensional Index

The choice of index dimensions depends on knowing the data and on analyzing what is needed from it. Examine the record selection (IF and WHERE) tests in your queries to see how many index dimensions each application actually uses to select records. If different applications need different subsets of dimensions, consider separate MDIs for the separate subsets. Although FOCUS can produce high-speed reporting performance with indexes of up to 30 dimensions, smaller indexes usually generate less retrieval overhead. You can create an unlimited number of MDIs.

The following are good candidates for dimensions in an MDI:

- Fields used frequently in record selection tests. Multiple fields used in selection tests within one query belong in the same MDI.
- Fields used as the basis for vertical partitioning, such as date or region.
- Derived dimensions (DEFINE fields) that define a new category based on an existing category; for example, if your data source contains the field STATE but you need region for your analysis, you can use the STATE field to derive the REGION dimension.
- Fields with many unique values. Fields which have few possible values are not normally good candidates. However, you may want to index such a field if the data source contains very few instances of one of the values, and you want to find those few instances.
- Packed decimal fields may be used as MDI dimensions on all platforms.
- An MDI can include a field that has a B-tree index as a dimension.
- MDI dimensions support missing values.

Including a field that is updated frequently (such as an AUTODATE field) in the MDI, requires frequent rebuilding of the MDI in order to keep it current. FOCUS can advise you on selecting MDI dimensions.

DEFINE fields described in the Master File can be used as dimensions. Dynamic DEFINE fields cannot be dimensions.

An MDI is for retrieval only; FIND and LOOKUP are not supported on an MDI.

Reference: Guidelines for a Multi-Dimensional Index

The following guidelines apply to each MDI:

- ❏ The maximum size of an MDI is 200 GB.
- ❏ The maximum size of each index partition is 2 GB.
- ❏ The total size of all dimensions in an MDI cannot exceed 256 bytes. However, if you include the MAXVALUES attribute in the Access File declaration for a dimension, FOCUS uses a small number of bytes to store the values of that dimension:

MAXVALUES	Number of Bytes Required
1 through 253	1
254 through 65,533	2
Greater than 65,533	4

To allow for expansion, if the maximum number of values is close to a limit, make MAXVALUES big enough to use a larger number of bytes. For example, if you have 250 values, specify 254 for MAXVALUES, and reserve 2 bytes for each dimension value.

Building and Maintaining a Multi-Dimensional Index

Example:

Creating a Multi-Dimensional Index in a FOCUS Session

Creating a Multi-Dimensional Index in a FOCEXEC

The REBUILD command is used to create or maintain a multi-dimensional index. This command can be issued in a FOCUS session or a FOCEXEC.

The best MDI is built by specifying the dimensions in order of best cardinality (most distinct values).

If issued in a FOCUS session, the REBUILD command conducts a dialogue with the user. To issue the REBUILD command in a FOCEXEC, you place the REBUILD command and the user-supplied information needed for REBUILD processing in the FOCEXEC.

If the MDI file might be larger than two gigabytes or if you plan to add more data partitions to it, the MDI index file must be partitioned from the initial REBUILD phase. After the index has been created, you can use it in a retrieval request. You cannot use an MDI for modifying the data source. If you update the data source without rebuilding the MDI and then attempt to retrieve data with it, FOCUS displays a message indicating that the MDI is out of date. You must then rebuild the MDI.

Example: Creating a Multi-Dimensional Index in a FOCUS Session

This example illustrates the creation of an MDI, CARMDI. The Access File contains the following:

```
MASTER = CAR,$
 DATA = 'CAR FOCUS A',$
 MDI = CARMDI,
   TARGET_OF = ORIGIN, $
   DIM = CAR,$
   DIM = COUNTRY,$
   DIM = MODEL,$
   MDIDATA = 'CARMDI MDI A',$
```

The top segment in the CAR data source, segment ORIGIN, is the TARGET_OF segment for the MDI. The following is the REBUILD procedure for the creation of the MDI.

1. Issue REBUILD at the command prompt.

   The REBUILD command invokes the REBUILD utility.

   The following menu displays:

   ```
   Enter option
   1. REBUILD (Optimize the database structure)
   2. REORG (Alter the database structure)
   3. INDEX (Build/modify the database index)
   4. EXTERNAL INDEX (Build/modify an external index database)
   5. CHECK (Check the database structure)
   6. TIMESTAMP (Change the database timestamp)
   7. DATE NEW (Convert old date formats to smartdate formats)
   8. MDINDEX (Build/modify a multidimensional index.)
   ```

2. Enter MDINDEX. The MDINDEX command invokes the MDI option of the REBUILD utility.

   The following message appears:

   ```
   NEW/ADD
   ```

3. Enter NEW. The NEW/ADD option enables you to create a new MDI or to add new data partitions to an existing MDI.

   The following message appears:

   ```
   ENTER THE NAME OF THE MASTER
   ```

4. Enter CAR. The CAR Master File contains the segment that is the TARGET_OF the MDI.

   The following message appears:

   ```
   ENTER MD_INDEX LOCATION FILE NAME
   ```

5. Enter CARMDI. CARMDI is the logical name of the CARMDI MDI.

   The following message appears:

   ```
   ANY RECORD SELECTION TESTS? (YES/NO)
   ```

6. Enter NO to indicate that there are no record selection criteria for the index. The user can limit the index to records that meet certain criteria by entering YES, followed by valid record selection expressions, followed by the END command.

Example: Creating a Multi-Dimensional Index in a FOCEXEC

The following FOCEXEC creates the CARMDI MDI and contains each user-supplied value needed for REBUILD processing on a separate line of the FOCEXEC, entered in uppercase:

```
REBUILD
MDINDEX
NEW
CAR
CARMDI
NO
```

Using a Multi-Dimensional Index in a Query

FOCUS allows you to use an MDI in a TABLE or SQL query. The performance is best when all of the dimensions in the MDI are used in selection criteria for the query.

There are two ways to use an MDI with a TABLE query:

- Lazy OR parameters. For example:

  ```
  IF (or WHERE) field EQ value_1 OR value_2 OR value_3. . .
  ```

- Mask or wildcard characters. For example, the following would show all values for a field that begin with A:

  ```
  IF (or WHERE) field EQ A*
  ```

  where *field* is a dimension in an MDI.

You can use an MDI with an SQL query by issuing an SQL SELECT statement with a WHERE test using a field of an MDI. For example,

```
SELECT field_1, field_2 FROM table WHERE field_3 = value;
```

where *field_3* is a dimension in an MDI.

Note: AUTOINDEX must be turned on for this feature to be operational.

Querying a Multi-Dimensional Index

How to:

Query a Multi-Dimensional Index

FOCUS provides a query command that generates statistics and descriptions for your MDIs. The command ? MDI allows you to display information about MDIs for a given FOCUS/XFOCUS Master File that hosts the target of your MDI.

Syntax: How to Query a Multi-Dimensional Index

```
? MDI masternamе {mdiname|*} [HOLD [AS holdfile]]
```

where:

masternamе

Is the logical name of the Master File. If you do not include any other parameters, a list of all MDI names specified is displayed with the command TARGET_OF in the Access File for this *masternamе*. If the Access File for the *masternamе* does not have any MDI information, a message will display.

mdiname

Is the logical name of an MDI. Specifying this parameter displays all the dimensions that are part of this MDI.

mdiname must be specified as TARGET_OF in the Access File for this *masternamе*, or a message will display. If any of the dimensions are involved in a parent-child structure, a tree-like picture will display.

*

Displays a list of all dimensions, by MDI, whose targets are specified inside the Access File for this *masternamе*.

HOLD

Saves the output in a text file.

holdfile

Is the file in which the output is saved. If this is not included with the AS phrase, the file is named HOLD.

Using AUTOINDEX to Choose an MDI

How to:

Enable or Disable AUTOINDEX for Retrieval

When an Access File defines multiple MDIs, retrieval efficiency for a query may become a function of the index it uses to access the data source. The AUTOINDEX facility analyzes each retrieval request and chooses the MDI or B-tree index that provides the best access to the data. You can issue the AUTOINDEX command at the command prompt, in a FOCEXEC, or in a profile. The AUTOINDEX facility can be enabled or disabled. The default is to disable AUTOINDEX, but FOCUS ships with a FOCPARM profile that turns it on when you enter FOCUS.

In its analysis, AUTOINDEX considers the following factors:

- The segment most involved in the query.
- The MDI with the most filtering conditions (IF/WHERE selection tests).
- The percent of index dimensions involved in the request from each MDI.
- How close the fields being retrieved are to the target segment.
- The size of each MDI.

If the selection criteria in a request do not involve any MDI fields, FOCUS looks for an appropriate B-tree index to use for retrieval. If a field is both a B-tree index and a dimension in an MDI, the MDI is used for retrieval if two-thirds of the fields in selection tests are dimensions in the MDI. If it is less than two-thirds, the B-tree index is used. If there are multiple B-tree indexes, the one highest in the hierarchy is used.

If everything else is equal, FOCUS uses the first MDI it finds in the Access File.

Syntax: How to Enable or Disable AUTOINDEX for Retrieval

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

where:

`ON`

Optimizes MDI retrieval. AUTOINDEX can only be set to ON when the ALL parameter is set to OFF. This value is not the default. However, the FOCPARM file shipped with FOCUS turns AUTOINDEX ON.

`OFF`

Disables the AUTOINDEX facility. No MDI will be used for retrieval. This value is the default.

Note: TABLE requests can automatically turn off AUTOINDEX and select the appropriate index for retrieval by indicating the index name in the request itself:

```
TABLE FILE filename.mdiname
```

You can also assure access with a specific MDI by creating an Access File that describes only that index.

Joining to a Multi-Dimensional Index

How to:

Join to All Dimensions of a Multi-Dimensional Index

Create a Dimensional Join

Reference:

Guidelines for Choosing a Source Field (Dimensional Joins Only)

Joining to an MDI uses the power of the MDI to produce a fast, efficient join. Instead of joining one field in the source file to an indexed field in the target file, you can join to multiple dimensions of the MDI.

When the join is implemented, the answer set from the source file is created, and the values retrieved for the source fields serve as index values for the corresponding dimensions in the target MDI.

You can join to an MDI in two ways:

- Join to all dimensions of a named MDI (MDI join). In the MDI join, you pick the MDI to use in the join.
- Join certain dimensions in a dimensional join. In this type of join, the JOIN engine picks the most efficient MDI based on the dimensions you choose. The dimensional join supports partial joins.

The source fields must form a one-to-one correspondence with the target dimensions. The MDI engine uses the source field values to get pointers to the target segment of the MDI, expediting data retrieval from the target file.

You can think of the source fields as mirror dimensions. If you put tighter constraints on the mirror dimensions, a smaller answer set is retrieved from the source file, and fewer index I/Os are required to locate the records from the target file. The speed of the join improves dramatically with the speed of retrieval of the source file answer set. Therefore, you can expedite any TABLE request against the source file by incorporating selection criteria on fields that participate in either a B-tree or MDI.

The following formula computes the time for a TABLE request that uses an MDI Join:

```
Total Time = Time to Retrieve the answer set from the source file (Ts)
+ Time to retrieve the MD index pointers (Tp)
+ Time to retrieve data from the target file (Tt)
```

Using a B-tree index or MDI in data retrieval reduces all types of retrieval time, reducing the total retrieval time.

Syntax: How to Join to All Dimensions of a Multi-Dimensional Index

```
JOIN field_1 [AND field_2 ...] IN sfile [TAG tag_1]
TO ALL mdiname IN tfile [TAG tag_2] [AS joinname]
[END]
```

where:

field_1, field_2

Are the join fields from the source file.

sfile

Is the source Master File.

tag_1, tag_2

Are one- to eight-character names that can serve as unique qualifiers for field names in a request.

mdiname

Is the logical name of the MDI, built on *tfile*, to use in the join.

tfile

Is the target Master File.

joinname

Is a one- to eight-character join name. A unique join name prevents a subsequent join from overwriting the existing join, allows you to selectively clear the join, and serves as a prefix for duplicate field names in a recursive join.

END

Is required to terminate the JOIN command if it is longer than one line.

Syntax: How to Create a Dimensional Join

```
JOIN field_1 [AND field_2 ...] IN sfile [TAG tag_1]
TO ALL dim_1 [AND dim_2 ...] IN tfile [TAG tag_2] [AS joinname]
[END]
```

where:

field_1, field_2

Are the join fields from the source file.

sfile

Is the source Master File.

tag_1, tag_2

Are one- to eight-character names that can serve as unique qualifiers for field names in a request.

dim_1, dim_2

Are dimensions in *tfile*.

tfile

Is the target Master File.

joinname

Is a one- to eight-character join name. A unique join name prevents a subsequent join from overwriting the existing join, allows you to selectively clear the join, and serves as a prefix for duplicate field names in a recursive join.

END

Is required to terminate the JOIN command if it is longer than one line.

Reference: Guidelines for Choosing a Source Field (Dimensional Joins Only)

- ❏ A maximum of four mirror and MDI dimensions can participate in a JOIN command.
- ❏ The target of the MDI must be a real segment in the target file.
- ❏ The order of the mirror dimensions must match the exact physical order of the MDI (target) dimensions.
- ❏ The format of each mirror dimension must be identical to that of the corresponding MDI dimension.
- ❏ The ALL attribute is required.

Encoding Values in a Multi-Dimensional Index

How to:

Retrieve Output From a Multi-Dimensional Index

Encode a Multi-Dimensional Index

Example:

Encoding a Multi-Dimensional Index

Using a Multi-Dimensional Index in a Request

Reference:

Rules for Encoding a Multi-Dimensional Index

FOCUS encodes indexed values any time a field or dimension of an MDI has a MAXVALUES attribute specified or is involved in a parent-child relationship. Encoded values are stored in the MDI file at rebuild time and can be retrieved and decoded with a TABLE request that specifies the MDIENCODING command. The MDIENCODING command allows the user to get output from the MDI file itself without having to read the data source.

Reference: Rules for Encoding a Multi-Dimensional Index

The following two rules apply to fields in a TABLE request that uses MDIENCODING:

- Only one MDI can be referred to at a time.
- Only dimensions that are part of the same parent-child hierarchy can be used simultaneously in a request. A dimension that is not part of a parent-child relationship can be used as the field in a request if it has a MAXVALUES attribute.

Syntax: How to Retrieve Output From a Multi-Dimensional Index

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

where:

`ON`

Enables retrieval of output from the MDI file without reading the data source.

`OFF`

Requires access of the data source to allow retrieval of MDI values.

Syntax: How to Encode a Multi-Dimensional Index

```
TABLE FILE mastername.mdiname
request
ON TABLE SET MDIENCODING ON
END
```

where:

mastername

Is the Master File.

mdiname

Is the logical name of the MDI.

request

Is the TABLE request that decodes the MDI.

Example: Encoding a Multi-Dimensional Index

The following examples show correct MDI encoding:

```
TABLE FILE COMPANY.I DATA1
PRINT CITY BY STATE
ON TABLE SET MDIENCODING ON
END
```

```
TABLE FILE COMPANY.I DATA1
COUNT CITY
IF STATE EQ NY
ON TABLE SET MDIENCODING ON
END
```

```
TABLE FILE COMPANY.I DATA1
PRINT CATEGORY
ON TABLE SET MDIENCODING ON
END
```

The following example is incorrect because CATEGORY is not part of the CITY-STATE hierarchy.

```
TABLE FILE COMPANY.I DATA1
PRINT CITY BY STATE
IF STATE EQ NY
IF CATEGORY EQ RESTAURANT
ON TABLE SET MDIENCODING ON
END
```

Example: Using a Multi-Dimensional Index in a Request

The following TABLE request accesses the CAR data source. It will use the CARMDI index for retrieval because CARMDI is the only MDI described in the Master File:

```
TABLE FILE CAR
SUM RETAIL_COST DEALER_COST
BY BODYTYPE
-* WHERE Condition utilizing MDI fields:
WHERE (COUNTRY EQ 'JAPAN' OR 'ENGLAND')
AND (CAR EQ 'TOYOTA' OR 'JENSEN' OR 'TRIUMPH')
AND (MODEL EQ 'COROLLA 4 DOOR DIX AUTO'
OR 'INTERCEPTOR III' OR 'TR7')
END
```

Partitioning a Multi-Dimensional Index

Example:

Adding a Partition to a Multi-Dimensional Index at the Command Prompt

Adding a Partition to a Multi-Dimensional Index in a FOCEXEC

If the data source has grown due to the addition of new data partitions, and these partitions need to be added to the MDI, you must perform the following steps:

1. Update the Access File to include the new data partitions.
2. Verify that your MDI is partitioned. Remember that the ADD function of the REBUILD utility cannot be executed on a non-partitioned MDI.
3. Perform the REBUILD, MDINDEX, and ADD on the MDI.

Example: Adding a Partition to a Multi-Dimensional Index at the Command Prompt

The following procedure is a REBUILD dialogue:

1. Issue REBUILD at the command prompt. The REBUILD command invokes the REBUILD utility.

   The following menu displays:

   ```
   Enter option
   1. REBUILD (Optimize the database structure)
   2. REORG (Alter the database structure)
   3. INDEX (Build/modify the database index)
   4. EXTERNAL INDEX (Build/modify an external index database)
   5. CHECK (Check the database structure)
   6. TIMESTAMP (Change the database timestamp)
   7. DATE NEW (Convert old date formats to smartdate formats)
   8. MDINDEX (Build/modify a multidimensional index.)
   ```

2. Enter MDINDEX. This command invokes the MDI option of the REBUILD utility.

   The following message appears:

   ```
   NEW/ADD
   ```

3. Enter ADD. The NEW/ADD option enables you to create a new MDI or to add new data partitions to an existing MDI.

   The following message appears:

   ```
   ENTER THE NAME OF THE MASTER
   ```

4. Enter the name of the Master File. The Master File contains the segment that is the TARGET_OF the MDI.

   The following message appears:

   ```
   ENTER MD_INDEX LOCATION FILE NAME
   ```

5. Enter the logical name of the MDI.

   The following message appears:

   ```
   ANY RECORD SELECTION TESTS? (YES/NO)
   ```

6. Enter NO to indicate that there are no record selection criteria for the index. The user can limit the index to records that meet certain criteria by entering YES, followed by valid record selection expressions, followed by the END command.

Once the MDI is rebuilt to include the new data partitions, any retrieval query that uses the MDI will use the newly added data partitions within that MDI.

Example: Adding a Partition to a Multi-Dimensional Index in a FOCEXEC

The following FOCEXEC contains commands that add a partition to a multi-dimensional index named CARMDI defined on the CAR data source:

```
REBUILD
MDINDEX
ADD
CAR
CARMDI
NO
```

After the MDI is rebuilt to include the new data partitions, any retrieval query that uses the MDI will use the newly added data partitions within that MDI.

Querying the Progress of a Multi-Dimensional Index

How to:

Query the Progress of a Multi-Dimensional Index

Use the SET MDIPROGRESS command to view messages about the progress of your MDI build. The messages will show the number of data records accumulated for every *n* records inserted into the MDI as it is processed.

Syntax: How to Query the Progress of a Multi-Dimensional Index

```
SET MDIPROGRESS = {0|n}
```

where:

n

Is an integer greater than 1000, which displays a progress message for every *n* records accumulated in the MDI build. 100,000 is the default value.

0

Disables progress messages.

Displaying a Warning Message

How to:

Display a Warning Message

The SET MDICARDWARN command displays a warning message every time a dimension cardinality exceeds a specified value, offering you the chance to study the MDI build. When the number of equal values of a dimension's data reaches a specified percent, a warning message will be issued. In order for MDICARDWARN to be reliable, the data source should contain at least 100,000 records.

Note: In addition to the warning message, a number displays in brackets. This number is the least number of equal values for the dimension mentioned in the warning message text.

Syntax: How to Display a Warning Message

```
SET MDICARDWARN = n
```

where:

n

Is a percentage value from 0 to 50.

7 Defining a Join in a Master File

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 are treated as if they were part of a single structure from which you can report. This section describes how to define a join in a Master File for FOCUS, fixed-format sequential, and VSAM data sources. For information about whether you can define a join in a Master File to be used with other types of data sources, see the appropriate data adapter manual.

Topics:

- ❏ Join Types
- ❏ 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 Joins
- ❏ Joining to One Cross-Referenced Segment From Several Host Segments

Join Types

You can join two data sources in the following ways:

- **Dynamically using the JOIN command.** The join lasts for the duration of the 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 data sources. You can also use the JOIN command to join two data sources of any type, including a FOCUS data source 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 319. This is supported for FOCUS data sources only.
- **Dynamically within a Master File.** This method saves you the trouble of issuing the JOIN command every time you need to join the data sources 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 333.

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, after the design is stable, to change the frequently used joins to static joins defined in the Master File, accelerating data source access. Static joins should be used when the target or cross-referenced data source contents do not change. You can change dynamic joins to static joins by using the REBUILD facility.

Note: Master File defined joins are sometimes referred to as cross-references.

Static Joins Defined in the Master File: SEGTYPE = KU and KM

In this section:

How to:

Example:

Static joins allow you to relate segments in different FOCUS data sources permanently. You specify static joins in the Master File of the host data source.

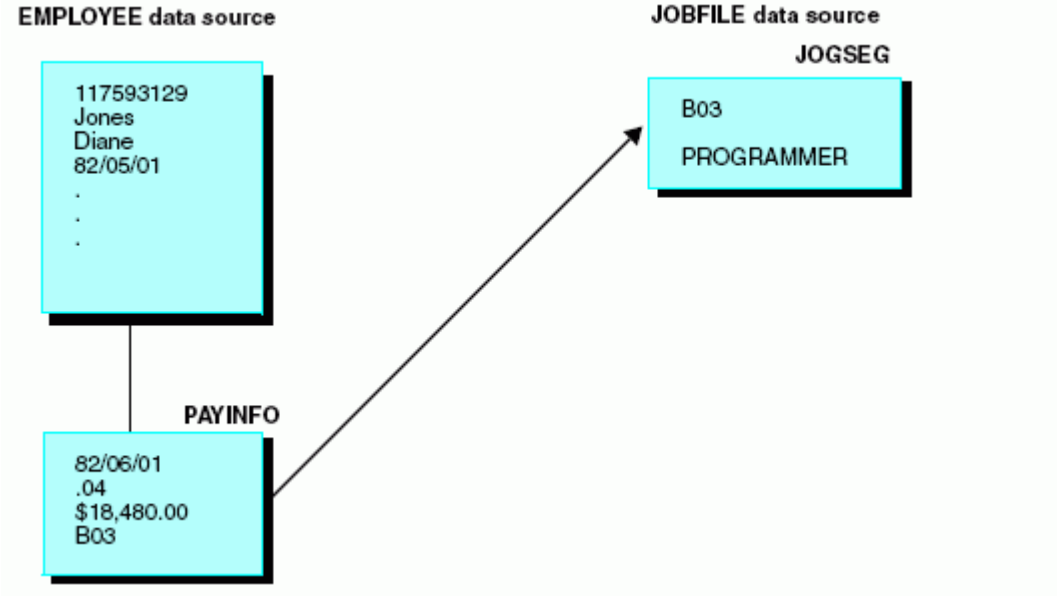
There are two types of static joins: one-to-one (SEGTYPE KU) and one-to-many (SEGTYPE KM).

- 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 data source for each record instance in the host data source.
- You specify a one-to-many join when you want to retrieve any number of record instances from the cross-referenced data source.

Describing a Unique Join: SEGTYPE = KU

In the EMPLOYEE data source, 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 data source named JOBFILE. You can retrieve the job description from JOBFILE by locating the record whose JOBCODE corresponds to the JOBCODE value in the EMPLOYEE data source, as shown in the following diagram:

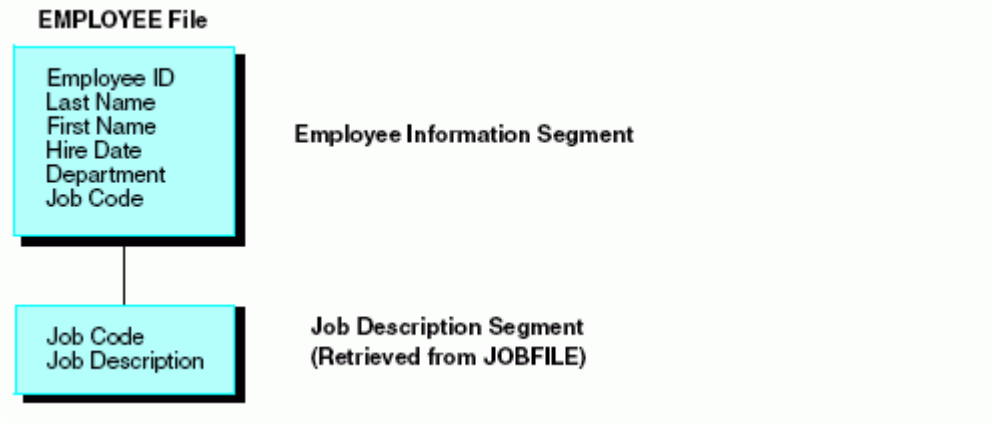


Using a join in this situation saves you the trouble of entering and revising the job description for every record in the EMPLOYEE data source. Instead, you can maintain a single list of valid job descriptions in the JOBFILE data source. Changes need be made only once, in JOBFILE, and are reflected in all of the corresponding joined EMPLOYEE data source 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 data sources, for reporting purposes the EMPLOYEE data source is treated as though it also contains the Job Description segment from the JOBFILE data source. The actual structure of the JOBFILE data source is not affected.

The EMPLOYEE data source is viewed as follows:



Syntax: How to Specify a Static Unique Join

```
SEGNAME = segname, SEGTYPE = KU, PARENT = parent,
CRFILE = db_name, CRKEY = field, [CRSEGNAME = crsegname,]
[DATASET = physical_filename,] $
```

where:

segname

Is the name by which the cross-referenced segment will be known in the host data source. You can assign any valid segment name, including the segment's original name in the cross-referenced data source.

parent

Is the name of the host segment.

db_name

Is the name of the cross-referenced data source. You can change the name without rebuilding the data source.

field

Is the common name (field name 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 data source 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 data source, you cannot change the crsegname without rebuilding the data source.

physical_filename

Optionally, is the platform-dependent physical name of the data source for the CRFILE.

The SEGTYPE value KU stands for keyed unique.

Example: Creating a Static Unique Join

```
SEGNAME = JOBSEG, SEGTYPE = KU, PARENT = PAYINFO,
   CRFILE = JOBFILE, CRKEY = JOBCODE, $
```

The relevant sections of the EMPLOYEE Master File follow (nonessential fields and segments 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 data source'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 data source, as well as the data source itself, must be accessible whenever the host data source is used. There does not need to be any data in the cross-referenced data source.

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 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

How to:

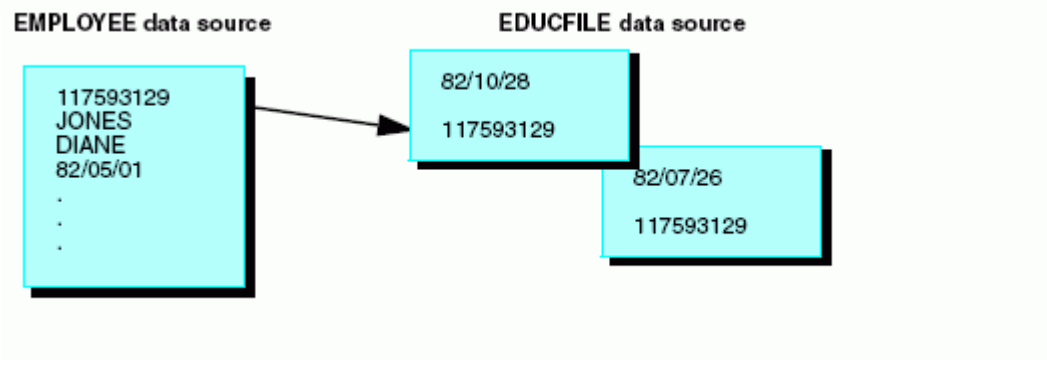
Specify a Static Multiple Join

Example:

Specifying a Static Multiple Join

You use a one-to-many join (that is, a non-unique join) when you may 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 data source named EDUCFILE to track course work for employees. One segment in that data source, 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 data source.

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 321, except that you supply a different value, KM (which stands for keyed multiple), for the SEGTYPE attribute, as follows:

```
SEGTYPE = KM
```

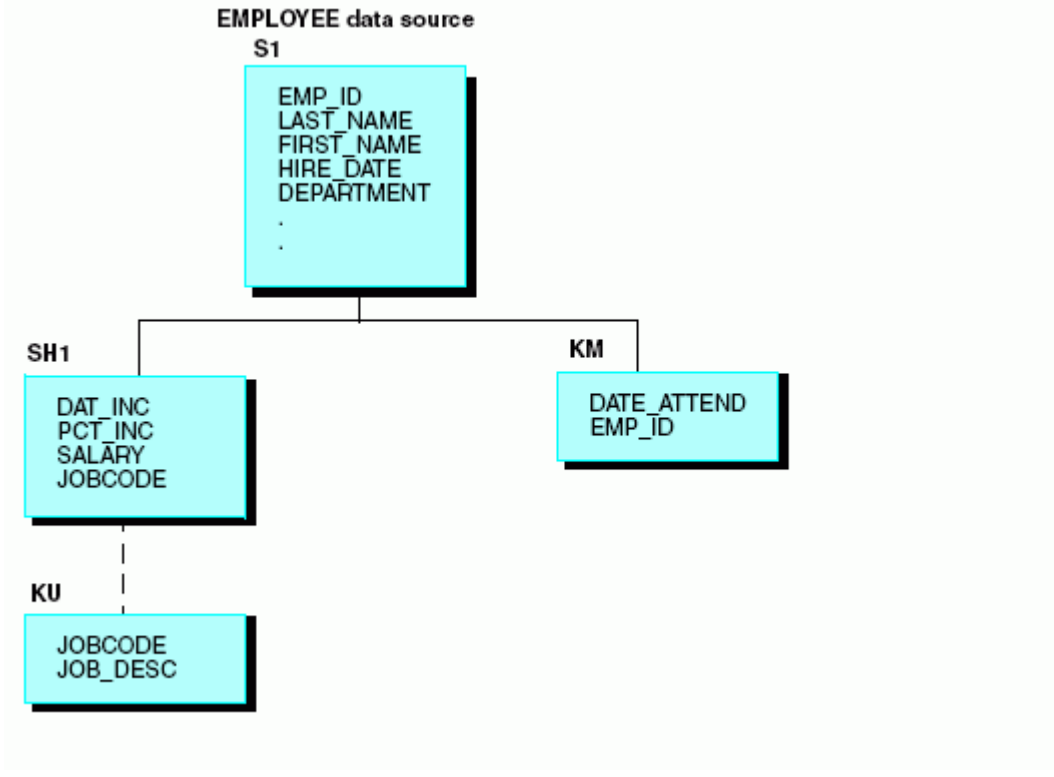
Example: Specifying a Static Multiple Join

```
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, both cross-referenced data sources, JOBFILE and EDUCFILE, are treated as though they are part of the EMPLOYEE data source. The data structure resembles the following:



Using Cross-Referenced Descendant Segments: SEGTYPE = KL and KLU

In this section:

Hierarchy of Linked Segments

How to:

Identify Cross-Referenced Descendant Segments

Example:

Identifying a Cross-Referenced Descendant Segment

Using a Cross-Referenced Descendant Segment

Using a Cross-Referenced Ancestral Segment

When you join two data sources, you can access any or all of the segments in the cross-referenced data source, not just the cross-referenced segment itself. These other segments are sometimes called linked segments. From the perspective of the host data source, 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 data source 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 data source.

Syntax: How to Identify Cross-Referenced Descendant Segments

```
SEGNAME = segname, SEGTYPE = {KL|KLU}, PARENT = parentname,
CRFILE = db_name, [CRSEGNAME = crsegname,]
[DATASET = physical_filename,] $
```

where:

segname

Is the name assigned to the cross-referenced segment in the host data source.

KL

Indicates that this segment is a descendant segment in a cross-referenced data source (as viewed from the perspective of the host data source), 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 data source (as viewed from the perspective of the host data source), 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 data source, as viewed from the perspective of the host data source.

db_name

Is the name of the cross-referenced data source. You can change the name without rebuilding the data source.

crsegname

Is the name of the cross-referenced segment. If you do not specify this, it defaults to the value assigned to SEGNAME.

physical_filename

Optionally, is the platform-dependent physical name of the data source for the CRFILE.

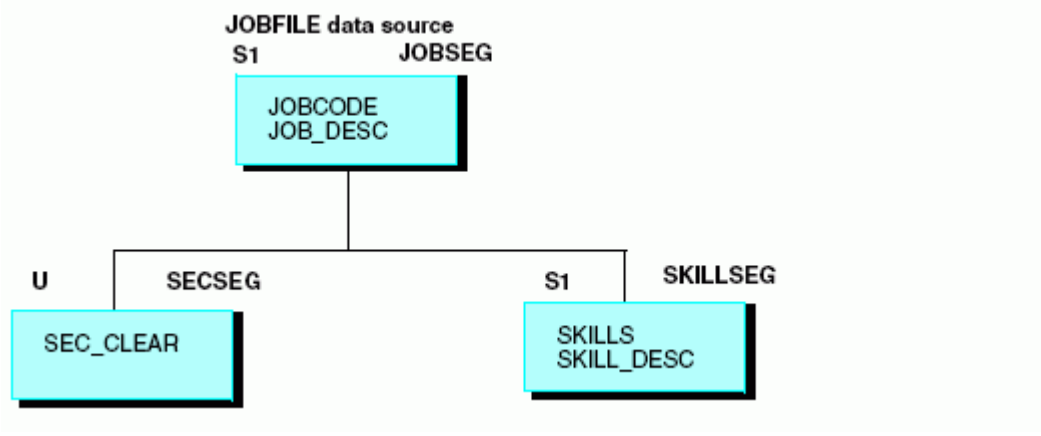
Example: Identifying a Cross-Referenced Descendant Segment

```
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) needs to be specified only for the cross-referenced segment.

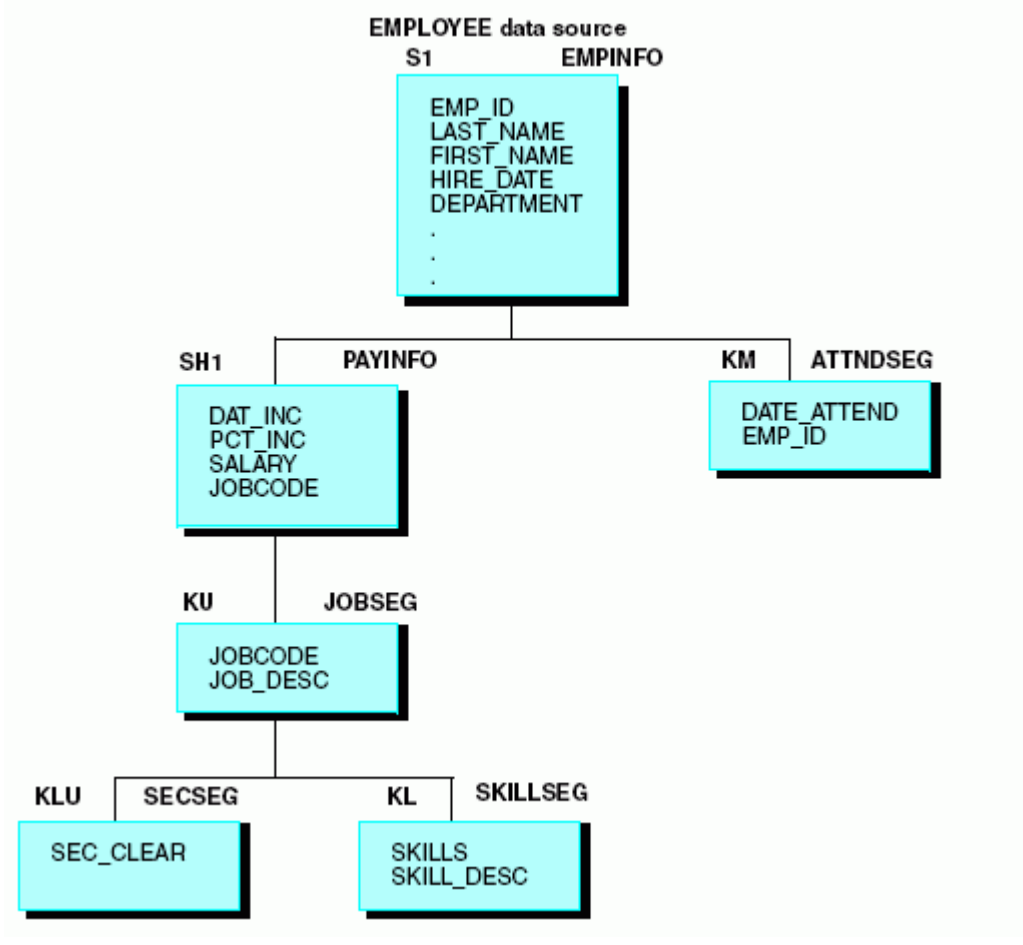
Example: Using a Cross-Referenced Descendant Segment

Consider our EMPLOYEE example. JOBFILE is a multi-segment data source:



In your EMPLOYEE data source application, you may need the security information stored in the SECSEG segment and the job skill information stored in the SKILLSEG segment. After you have created a join, you can access any or all of the other segments in the cross-referenced data source using the SEGTYPE value KL for a one-to-many relationship (as seen from the host data source), and KLU for a one-to-one relationship (as seen from the host data source). KL and KLU are used to access descendant segments in a cross-referenced data source for both static (KM) and dynamic (DKM) joins.

When the JOBSEG segment is retrieved from JOBFILE, it also retrieves all of JOBSEG's children that were declared with KL or KLU SEGTYPEs in the EMPLOYEE Master File:



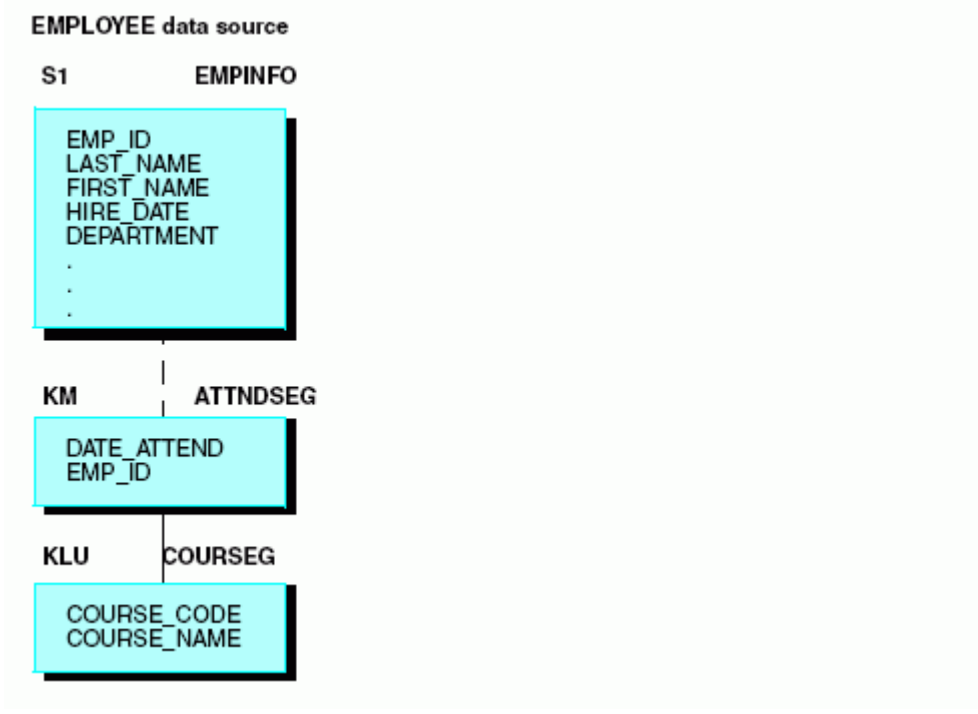
Example: Using a Cross-Referenced Ancestral Segment

Remember that you can retrieve all of the segments in a cross-referenced data source, 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 data source, this is a one-to-one relationship.

Consider the EDUCFILE data source 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, as shown in the following diagram.



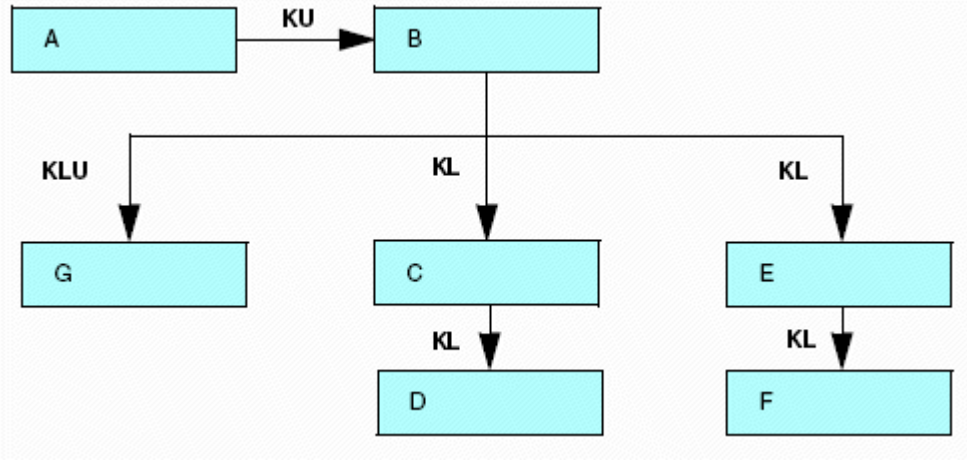
The sections of the EMPLOYEE Master File used in the examples follow (nonessential fields and segments are not shown).

```
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, $
```

Hierarchy 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

How to:

Specify a Dynamic Join in a Master File

Example:

Specifying a Dynamic Join in a Master File

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, 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 data source for each record instance in the host data source.
- You specify a one-to-many join when you want to retrieve any number of record instances from the cross-referenced data source.

The difference between static and dynamic joins deals with storage, speed, and flexibility:

- The links (pointers) for a static join are retrieved once and then permanently stored in the host data source (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 redefine or remove a static join only using the REBUILD facility. You can redefine or remove a dynamic join at any time by editing the Master File.

Syntax: How to Specify a Dynamic Join in a Master File

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 321), 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 in a Master File

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 Joins

To join two FOCUS data sources, 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 data source.
- For a dynamic join, the links are retrieved each time needed. This makes static joins faster than dynamic ones, since the links only need to be established once, but less flexible, as you can redefine or remove a static join only by using the REBUILD facility or reloading the file with MODIFY or Maintain.

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 data source. 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 **(SEGTYPE = KU or KM)**	Faster after first use; links are created only once. Always in effect. Can select some linked segments and omit others.	Must be specified before data source is created or reloaded using REBUILD. Requires REBUILD facility to change. Requires four bytes of file space per instance. User needs to know how to specify relationships for linked segments (KL, KLU).
Dynamic Join in Master File **(SEGTYPE = DKU or DKM)**	Can be specified at any time. Always in effect. Does not use any space in the data source. Can be changed or removed as needed, without using the REBUILD facility. Can select some linked segments and omit others.	Slower; links are retrieved for each record in each report request. 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. Does not use any space in the data source. Can be changed or removed as needed, without using the REBUILD facility. User never needs to describe relationships of linked segments.	Slower; links are retrieved for each record in each report request. JOIN command must be issued in each session in which you want the join to be in effect. All segments in the target are always included, whether or not you need them.

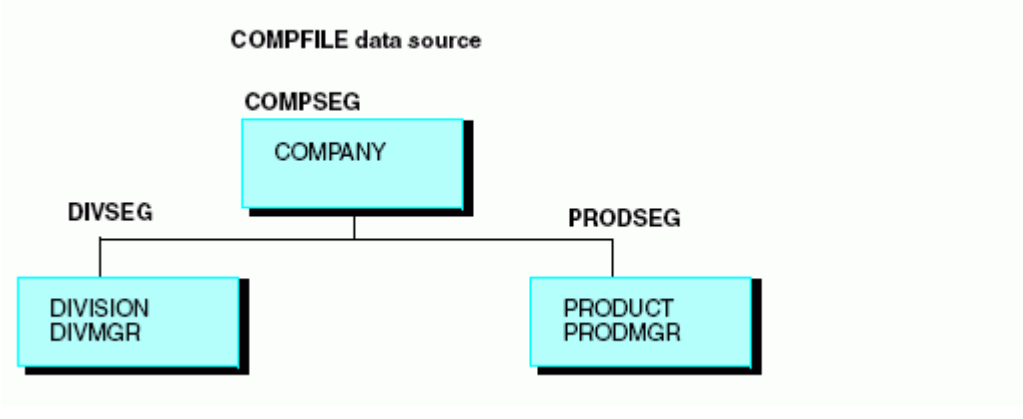
Joining to One Cross-Referenced Segment From Several Host Segments

In this section:

You may come upon situations where you need to join to one cross-referenced segment from several different segments in the host data source. You may also find a need to join to one cross-referenced segment from two different host data sources at the same time. You can handle these data structures using Master File defined joins.

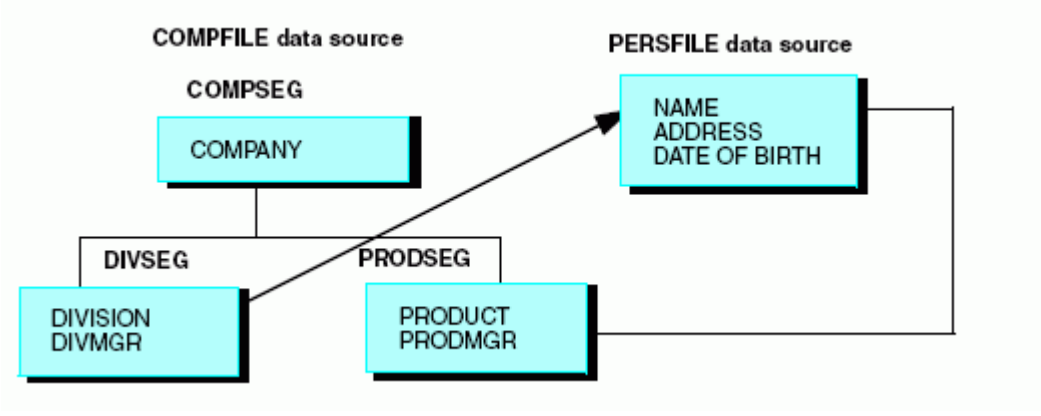
Joining From Several Segments in One Host Data Source

In an application, you may want to use the same cross-referenced segment in several places in the same data source. Suppose, for example, that you have a data source 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.

Retrieve personal information for both the product managers and the division managers from a single personnel data source, 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 data source (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.

A way is provided for you to implement a join to the same cross-referenced segment from several segments in the one host data source; 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 could 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. The common alias, NAME, is automatically matched to the field name NAME in the PERSFILE data source. In addition, the field declarations following the join information rename the ADDRESS and DOB fields to be referred to separately in reports. The actual field names in 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 data source. That it cannot be renamed is not a problem, since its ALIAS can be renamed and 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 data source must be identical to its FIELDNAME in the cross-referenced data source.
- ❏ 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.

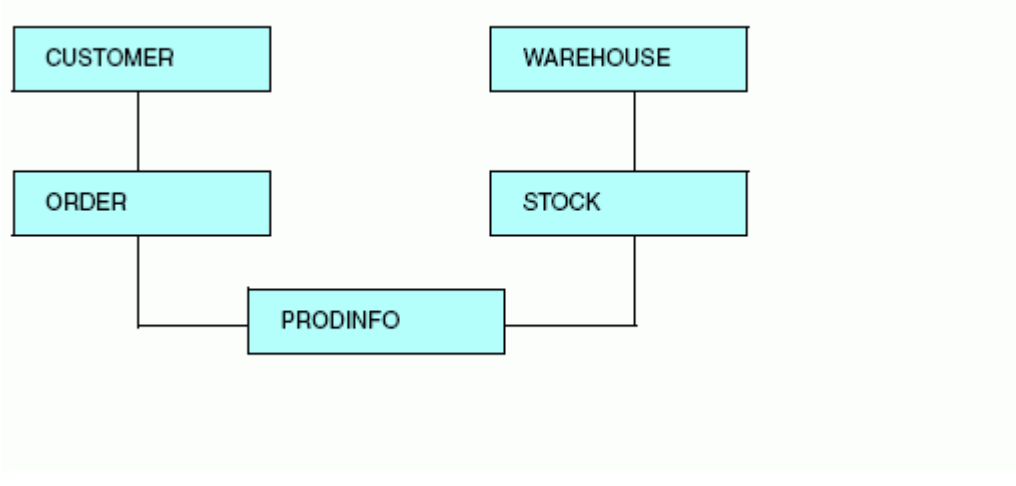
- Place field declarations for the cross-referenced segment after the cross-referencing information in the Master File of the host data source, in the order in which they actually occur in the cross-referenced segment. Omit the record terminator ($) at the end of the cross-referenced segment declaration 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          , $
```

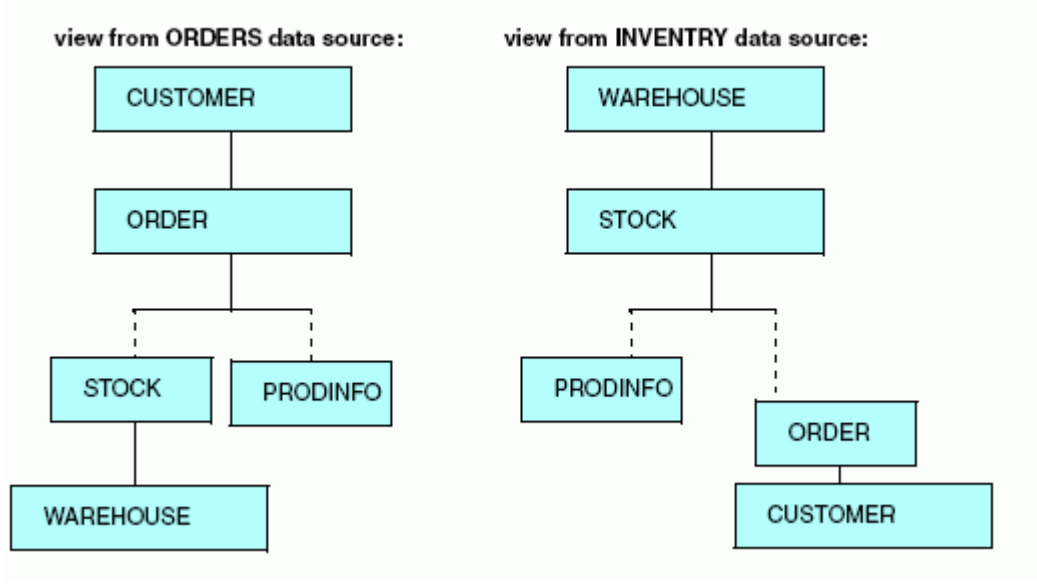
Joining From Several Segments in Several Host Data Sources: Multiple Parents

At some point you may need to join to a cross-referenced segment from two different host data sources at the same time. If you were to describe a structure like this as a single data source, you would have to have two parents for the same segment, which is invalid. You can, however, describe the information in separate data sources, 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 data source, it would be structured as follows:



You can join several data sources to create this structure. For example:



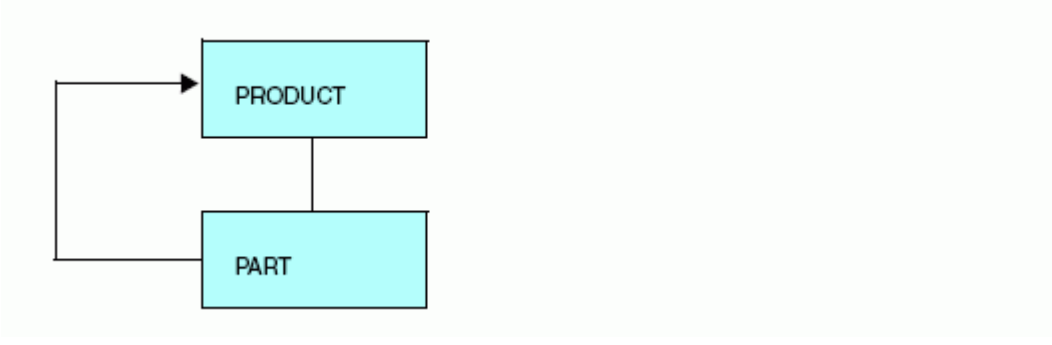
The CUSTOMER and ORDER segments are in the ORDERS data source, the WAREHOUSE and STOCK segments are in the INVENTRY data source, and the PRODINFO segment is stored in the PRODUCTS data source. Both the INVENTRY and ORDERS data sources have one-to-one joins to the PRODUCTS data source. In the INVENTRY data source, STOCK is the host segment; in the ORDERS data source, ORDER is the host segment.

In addition, there is a one-to-many join from the STOCK segment in the INVENTRY data source to the ORDER segment in the ORDERS data source, and a reciprocal one-to-many join from the ORDER segment in the ORDERS data source to the STOCK segment in the INVENTRY data source.

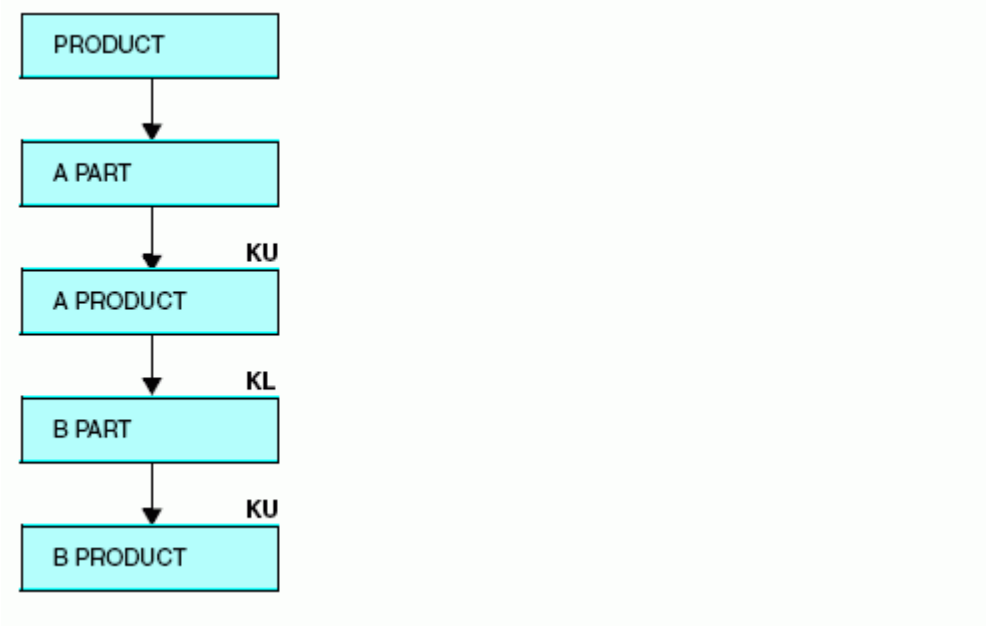
The joins among these three data sources can be viewed from the perspectives of both host data sources, approximating the multiple-parent structure described earlier.

Recursive Reuse of a Segment

In rare cases, a data source may cross-reference itself. Consider the case of a data source of products, each with a list of parts that compose the product, where a part may itself be a product and have subparts. Schematically, this would appear as:



A description for this case, shown for two levels of subparts, is:



See the *Creating Reports* manual for more information on recursive joins.

8 Checking and Changing a Master File: CHECK

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, your Master File may not be reloaded into memory if a parsed copy already exists.

Topics:

- ❏ Checking a Data Source Description
- ❏ CHECK Command Display
- ❏ PICTURE Option
- ❏ HOLD Option

Checking a Data Source Description

The CHECK output highlights any errors in your Master File and allows you to correct each 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 a Data Source Description

```
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 data source structure. The keyword PICTURE can be abbreviated to PICT. This option is explained in *PICTURE Option* on page 348.

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 the parent segment. The keyword RETRIEVE can be abbreviated to RETR.

DUPLICATE

Lists duplicate field names for the specified data source. The keyword DUPLICATE can be abbreviated to DUPL.

HOLD

Generates a temporary HOLD file and HOLD Master File containing information about fields in the data source. You can use this HOLD file to write reports. The AS option specifies a field name for your data sources. The option is described and illustrated in *HOLD Option* on page 351.

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

In this section:

Determining Common Errors

Example:

Using the CHECK File Command

If your Master File contains syntactical errors, the CHECK command displays appropriate messages.

If the data source 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 sources containing the fields.

NUMBER OF DEFINES

Is the number of virtual fields in the Master File. This message appears only if virtual fields are defined.

TOTAL LENGTH

Is the total length of all fields as defined in the Master File by either the FORMAT attribute (if the data source is a FOCUS data source) or the ACTUAL attribute (if the data source is a non-FOCUS data source).

Example: Using the CHECK File Command

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 using FOCUS online, this message displays on the terminal even if the PRINT parameter is set to OFFLINE.

Determining Common Errors

- If the data source is a non-FOCUS data source, check the TOTAL LENGTH OF ALL FIELDS to verify the accuracy of the field lengths specified for the data source. One of the most common causes of errors in generating reports from non-FOCUS data sources 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 non-FOCUS data source.

  In general, if the total length of all fields is not equal to the logical record length of the non-FOCUS data source, 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.

  When the CHECK command is issued for a data source with more than one field of the same name in the same segment, a FOC1829 message is generated along with a warning such as the following indicating that the duplicate fields ncannot be accessed:

  ```
  (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 that indicates where the first duplicate field resides:

  ```
  WARNING: FOLLOWING FIELDS APPEAR MORE THAN ONCE
  AA   IN SEGMENT SEGB         (VS SEGA)
  ```

PICTURE Option

Example:

Using the CHECK FILE PICTURE Option

The PICTURE option displays a diagram of the 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 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 greater than 12 characters are truncated to 12 characters in CHECK FILE PICTURE operations, with the last character appearing as a ' >', indicating that more characters exist than can be shown.

I

Indicates an indexed field.

K

Indicates the key field in the cross-referenced segment.

crfile

Is the name of the cross-referenced data source 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 data source joined to the SALARY data source:

```
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 01/31/03 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
```

HOLD Option

In this section:

Example:

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 the FDEFCENT and FYRTHRESH 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 structure. This appears in the picture.

LEVEL

The level of the segment within the 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, do 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 procedure creates a HOLD file describing the EMPLOYEE data source. It then writes a report that displays the names of cross-referenced segments in the EMPLOYEE data source, the segment types, and the attributes of the 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 DATA SOURCE"
" "
PRINT FIELDNAME/A12 ALIAS/A12 USAGE BY SEGNAME BY SEGTYPE
WHERE SEGTYPE CONTAINS 'K'
END
```

The output is:

```
PAGE 1

FIELDNAMES, ALIASES, AND FORMATS
OF CROSS-REFERENCED FIELDS IN THE EMPLOYEE DATA SOURCE

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 data source 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 output:

```
FDEFCENT    FYRTHRESH
--------    ---------
      19           50
```

Specifying an Alternate File Name 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 any existing default file will be replaced.

Note: 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 using the HOLD option of the CHECK command, the TITLE text is placed 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 data source 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 data sources.

Virtual Fields in the Master File

With the HOLD option, virtual fields are placed in the segment in which they would be stored if they were real fields in the data source. 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 appear as blank in reports. FLDSEG may be dynamically reformatted in a TABLE request (FLDSEG/I2) to force the display of zero.

After data has been entered into a data source, 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 data source 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 Master File. The checking procedure, CHECK, should be used after any change.

9 Accessing a FOCUS Data Source: USE

The USE command specifies the names and locations of FOCUS data sources for the following conditions:

- Default naming conventions are not used.
- You need to protect data sources from change or concatenate several similar data sources.

Note:

- For non-default naming conventions, you may be able to use the DATASET attribute in the Master File instead of the USE command. For detailed information, see Chapter 2, *Identifying a Data Source*.
- You can use an Access File instead of a USE command to concatenate FOCUS data sources. When a FOCUS data source is partitioned, an Access File may be required for efficient retrieval. For more information, see Chapter 6, *Describing a FOCUS Data Source*.
- You must issue a USE command in order to work with an external index.

Topics:

USE Command

How to:

Issue a USE Command

Specify Multiple Data Sources

Erase USE Specifications

When you issue a FOCUS command to access a FOCUS data source, such as TABLE FILE *filename*, FOCUS searches for a Master File with the specified file name, and then searches for a data source with the same file name in CMS or allocated to the same ddname in z/OS.

In CMS FOCUS, the FOCUS data source has the default file type FOCUS and the default file mode A.

For example, the command TABLE FILE EMPLOYEE uses the data source EMPLOYEE FOCUS A for CMS or the data source allocated to ddname EMPLOYEE for z/OS.

If the FOCUS data source has a file ID different from these defaults, you must issue the USE command to identify the data source, with its file specifications, and associate it with a specific Master File.

The USE command specifies the names and locations of FOCUS data sources for the following conditions:

- Default naming conventions are not used.
- To protect data sources from change or concatenate several similar data sources.

When you identify FOCUS data sources with the USE command, a USE directory is created, which is a list of data source definitions. When a USE directory is in effect, FOCUS will locate data sources using the information in the directory, instead of searching for the data source using default names. A USE directory enables you to access up to 255 data sources. The USE directory applies only to FOCUS data sources.

Syntax: How to Issue a USE Command

```
USE action
fileid [READ|NEW] [AS masternamе]
```

or

```
fileid AS masternamе ON server READ
```

or

```
fileid LOCAL
```

or

```
fileid ON server
.
.
.
ind {WITH|INDEX} masternamе
END
```

where:

action

Is one of the following:

ADD appends one or more new file IDs to the present directory. If you issue the USE command without the ADD parameter, the list of data sources 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 file ID in the USE directory. This option enables you to change the file specification for the file ID and the options following the file ID.

fileid

Is any valid file name for the specific operating system.

Platform	File ID
CMS	Any valid file name, in *filename filetype filemode* format containing your FOCUS data source.
z/OS	A *ddname* allocated to the z/OS data set containing your FOCUS data source.

READ

Restricts data sources to read-only access.

`NEW`

Indicates that the data source has yet to be created.

`AS` *`mastername`*

Specifies the name of the Master File to be associated with the file ID.

`ON` *`server`*

Specifies the userid of the FOCUS Database Server (sink machine) that synchronizes FOCUS data sources for use by multiple users on CMS, or the communications data set of the FOCUS Database Server on z/OS.

`LOCAL`

This option requires a previous directory entry for the file ID with the ON *server* option. For CMS, accesses an SU data source directly through the operating system. Before using this option, you must link and access the minidisk on which the SU data source resides in read-only mode.

For z/OS, accesses an SU data source using the Multi-Threaded SU Reporting Facility. Before using this option, you must allocate the SU data source in SHR mode.

`ind`

Is the file ID (on CMS) or ddname (on z/OS) of an external index.

`WITH|INDEX`

Establishes the relationship between an external index and the component data source. INDEX is a synonym for WITH.

The following options after the file ID are valid together:

```
READ and AS
NEW and AS
AS and ON and READ
```

Any other combination of options after the file ID is not valid.

Syntax: How to Specify Multiple Data Sources

You can specify several data sources 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 USE Specifications

To erase the USE specifications, enter the following command:

```
USE CLEAR
```

Specifying a Non-Default File ID

How to:

Specify a Non-Default File ID

Example:

Specifying Different File Names

Specifying Different File Types and Extensions

Specifying Different File Locations

If the FOCUS data source has a file ID other than the default, issue the USE command to identify the data source, with its file specifications, and associate it with a specific Master File.

Syntax: How to Specify a Non-Default File ID

```
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 file ID.

Example: Specifying Different File Names

To read the data source with the name EMP026 described by the Master File EMPLOYEE, enter this USE command:

For CMS:

```
USE
EMP026 FOCUS A AS EMPLOYEE
END
```

For z/OS:

```
USE
EMP026 AS EMPLOYEE
END
```

After entering the USE command, you can read the EMP026 data source by entering the command TABLE FILE EMPLOYEE.

Example: Specifying Different File Types and Extensions

For CMS: To read the data source with the name EMP026 and a file type of DATA on the A-disk, described by the Master File EMPLOYEE, enter this USE command:

```
USE
EMP026 DATA A AS EMPLOYEE
END
```

After entering the USE command, you can read the EMP026 data source by entering the command TABLE FILE EMPLOYEE.

Example: Specifying Different File Locations

For CMS: To read the data source 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 data source 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 data source PRODUCT FOCUS F even if you did not specify the data source 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 Data Sources to FOCUS

How to:

Identify New Data Sources to FOCUS

Example:

Identifying a New Data Source

The parameter NEW in the USE command identifies data sources that do not exist yet. When you identify a new data source, you can accept the default file specification conventions or specify different ones.

In CMS, when you issue a MODIFY command specifying a data source that does not exist, FOCUS creates the data source with a default file name, file type, and file mode. You can issue the USE command with the NEW parameter to give the data source a file ID other than the default.

For z/OS, you must allocate the data source, with the z/OS command ALLOCATE or the FOCUS command DYNAM, before you issue the USE command.

Syntax: How to Identify New Data Sources to FOCUS

```
USE
fileid NEW
END
CREATE file mastername
```

where:

fileid

Is any valid file specification for the operating system. The file ID will be assigned to the data source later in the session when the actual create happens.

On z/OS, the FOCUS data source will be created as a temporary file unless you previously allocated a permanent data source to the ddname in the USE command.

mastername

Is the name of the Master File associated with the data source.

If you omit the NEW parameter, a message is returned stating that the data source cannot be found, and the USE command is not executed.

Example: Identifying a New Data Source

To create the data source WAGES using the WAGES Master File, enter the following:

For CMS:

```
USE
WAGES FOCUS F NEW
END
CREATE FILE WAGES
```

For z/OS:

```
USE
WAGES NEW
END
CREATE FILE WAGES
```

Accessing Data Sources in Read-Only Mode

How to:

Access a Data Source in Read-Only Mode

Example:

Accessing a Data Source in Read-Only Mode

You can protect data sources from changes by issuing USE commands with the READ parameter. Protected data sources can be read by various FOCUS tools and commands such as MODIFY and SCAN, but cannot be changed.

Syntax: How to Access a Data Source in Read-Only Mode

```
USE
fileid READ
END
```

where:

fileid

Is any valid file specification for the operating system.

Example: Accessing a Data Source in Read-Only Mode

For example, to protect the data source EMPLOYEE, enter:

For CMS:

```
USE
EMPLOYEE FOCUS A READ
END
```

For z/OS:

```
USE
EMPLOYEE READ
END
```

Concatenating Data Sources

How to:

Concatenate Data Sources

Example:

Concatenating Data Sources to One Master File

Concatenating Multiple Master Files

Concatenating Multiple Data Sources and a Single Cross-Reference Data Source

Concatenating Multiple Data Sources and Multiple Cross-Reference Data Sources

If several FOCUS data sources are described by the same Master File, you can read all of the data sources in one TABLE or GRAPH request by issuing a USE command that concatenates all of the data sources.

Syntax: How to Concatenate Data Sources

```
USE
fileid1 AS mastername
fileid2 AS mastername
.
.
fileidn AS mastername
END
```

where:

fileid1...

Are any valid file specifications, for the operating system, for the data sources being concatenated.

mastername

Is the name of the Master File that describes the data sources.

Example: Concatenating Data Sources to One Master File

For example, to read three FOCUS data sources: 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 z/OS:

```
USE
EMP024 AS EMPLOYEE
EMP025 AS EMPLOYEE
EMP026 AS EMPLOYEE
END
```

You can then read all three data sources with the command TABLE FILE EMPLOYEE.

Example: Concatenating Multiple Master Files

You can concatenate data sources to several Master Files in one USE command. For example, the following USE command concatenates the EMP01 and EMP02 data sources to the Master File EMPLOYEE, and concatenates the SALES01 and SALES02 data sources 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 z/OS:

```
USE
EMP01 AS EMPLOYEE
EMP02 AS EMPLOYEE
SALES01 AS SALES
SALES02 AS SALES
END
```

To read the EMP01 and EMP02 data sources, begin by entering

```
TABLE FILE EMPLOYEE
```

and to read the SALES01 and SALES02 data sources, begin by entering

```
TABLE FILE SALES
```

Example: Concatenating Multiple Data Sources and a Single Cross-Reference Data Source

To read multiple data sources with a cross-reference data source as one data source, specify the host data sources in the USE command and then the cross-reference data source.

For example, the data source EMPLOYEE is made up of two data sources EMP01 and EMP02 that reference a common cross-reference data source EDUCFILE. To read the two data sources together, enter the following USE command:

For CMS:

```
USE
EMP01 FOCUS A AS EMPLOYEE
EMP02 FOCUS A AS EMPLOYEE
EDUCFILE FOCUS A
END
```

For z/OS:

```
USE
EMP01 AS EMPLOYEE
EMP02 AS EMPLOYEE
EDUCFILE
END
```

Example: Concatenating Multiple Data Sources and Multiple Cross-Reference Data Sources

If the EMPLOYEE data source consisted of two data sources, EMP01 and EMP02, and each had its own cross-reference data source, ED01 and ED02, you can read all four data sources in one command by entering this USE command where each host data source is followed by its cross-reference.

You cannot specify a concatenated data source as the cross-referenced data source in a JOIN command.

Take an indexed view of a concatenated data source by creating an external index data source and using the TABLE FILE *filename.indexed_fieldname* command. For more information about indexed views, see the instructions for creating an external index data source 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 z/OS:

```
USE
EMP01 AS EMPLOYEE
ED01  AS EDUCFILE
EMP02 AS EMPLOYEE
ED02  AS EDUCFILE
END
```

Accessing Simultaneous Usage Data Sources

In this section:

Multi-Thread Configuration

How to:

Access Simultaneous Usage Data Sources

Read SU Data Sources in a Multi-Thread Configuration

Example:

Accessing an SU Data Source in CMS

Accessing an SU Data Source in z/OS

In CMS, the FOCUS Database Server is a disconnected virtual machine that manages all READ/WRITE operations to a FOCUS data source.

In z/OS, the FOCUS Database Server is a batch job or started task managing all READ/WRITE operations to a FOCUS data source.

Syntax: How to Access Simultaneous Usage Data Sources

```
USE
fileid ON server
END
```

where:

fileid

In CIMS, is the file name, file type, and file mode of the FOCUS data source accessed by the disconnected virtual machine (FOCUS Database Server).

In z/OS, is lthe ddname of the FOCUS data source allocated in the batch job or started task.

Example: Accessing an SU Data Source in CMS

To use the EMPLOYEE FOCUS data source 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 Data Source in z/OS

To use the EMPLOYEE FOCUS data source allocated to the FOCUS Database Server batch job or started task, two things must be done:

1. Allocate a ddname to the communications data set that is allocated in the FOCUS Database Server batch job or started task pointed to by the ddname FOCSU.

   For example,

   ```
   DYNAM ALLOC FILE MYSERVER DS prefix.FOCSU.DATA SHR
   ```

2. Issue the USE command for your data source allocated in the batch job or started task. For example:

   ```
   USE
   EMPLOYEE ON MYSERVER
   END
   ```

Multi-Thread Configuration

Performance gains may be achieved by routing read-only requests directly to the data source on disk instead of through the FOCUS Database Server. This is called a multi-thread configuration. It is accomplished with the USE command and the keyword LOCAL.

Syntax: How to Read SU Data Sources in a Multi-Thread Configuration

In CMS, first link and access the data source 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 z/OS, allocate the FOCUS data source 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 your Simultaneous Usage documentation.

Note:

- On a FOCUS Database Server, 255 data sources can be open at one time with 256 users connected.
- On VM, the Multi-Threaded SU option requires filemode 6.
- The READ option is available for accessing an SU data source 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. It is required because alternate Master Files are not supported in SU for MODIFY and MAINTAIN.

Using the LOCATION Attribute

The file type and physical location of a data source named by the LOCATION attribute in the Master File defaults to FOCUS in the local directory unless a USE command is issued or the DATASET attribute is specified in the Master File. If the physical data sources 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 data sources 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
```

10 Providing Data Source Security: DBA

As Database Administrator, you can use DBA security features to provide security for any FOCUS data source. You can use these security features to limit the number of records or reads a user can request in a report.

You can also create user-written programs to perform program accounting on FOCUS data sources. 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 DBA security features to provide security for non-FOCUS data sources. However, the RESTRICT command is not available for those data sources. Note that DBA security cannot protect a data source from non-FOCUS access.

Note: All references to FOCUS data sources also apply to XFOCUS data sources.

Topics:

Introduction to Data Source Security

The DBA facility provides a number of security options:

- You can limit the users who have access to a given data source using the USER attribute discussed in *Identifying Users With Access Rights: The USER Attribute* on page 379.
- You can restrict a user's access rights to read, write, or update only using the ACCESS attribute discussed in *Specifying an Access Type: The ACCESS Attribute* on page 385.
- You can restrict a user's access to certain fields or segments using the RESTRICT attribute discussed in *Limiting Data Source Access: The RESTRICT Attribute* on page 390.
- You can ensure that only records that pass a validation test are retrieved using the RESTRICT attribute discussed in *Limiting Data Source Access: The RESTRICT Attribute* on page 390.
- You can limit the values a user can read or write to the data source or you can limit which values a user can alter using the RESTRICT attribute discussed in *Limiting Data Source Access: The RESTRICT Attribute* on page 390.
- You can point to passwords and restrictions stored in another Master File with the DBAFILE attribute discussed in *Placing Security Information in a Central Master File* on page 399.
- You can use the FOCUSID exit routine to allow an external security system set the FOCUS password.
- You can place security on FOCEXECs, as discussed in *FOCEXEC Security* on page 409.

Program accounting, resource limitation, and the UACCT exit routine are discussed in *Program Accounting/Resource Limitation* on page 411.

Also, whenever a new data source 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 data source. The shadow paging feature is discussed in *Absolute File Integrity* on page 415.

Implementing Data Source Security

In this section:

Identifying the DBA: The DBA Attribute

Including the DBA Attribute in a HOLD File

Identifying Users With Access Rights: The USER Attribute

Non-Overridable User Passwords (SET PERMPASS)

Controlling Case Sensitivity of Passwords

Establishing User Identity

Example:

Implementing Data Source Security in a Master File

Reference:

Special Considerations for Data Source Security

You provide FOCUS security on a file-by-file basis. Implementing DBA security features is a straightforward process in which you specify:

- The names or passwords of FOCUS users granted access to a data source.
- The type of access the user is granted.
- The segments, fields, or ranges of data values to which the user's access is restricted.

The declarations (called security declarations) follow the END command in a Master File and tell FOCUS that security is needed for the data source and what type of security is needed. Each security declaration consists of one or several of the following attributes:

- The DBA attribute gives the name or password of the Database Administrator for the data source. The Database Administrator has unlimited access to the data source and its Master File.
- The USER attribute identifies a user as a legitimate user of the data source. Only users whose name or password is specified in the Master File of a FOCUS data source with security placed on it have access to that data source.

- The ACCESS attribute defines the type of access a given user has. The four types of access available are:

  RW, which allows a user to both read and write to a data source.

  R, which allows a user only to read data in a data source.

  W, which allows a user to only write new segment instances to a data source.

  U, which allows a user only to update records in a data source.

- 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.

Describe your data source security by specifying 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.

Example: Implementing Data Source Security in a Master File

The following is a Master File that uses 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,$
```

Reference: Special Considerations for Data Source Security

- ❏ When using the JOIN command, it is possible to bypass the DBA information in a data source. 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 Master File* on page 399. All data sources in the joined structure will get security information as coded in the DBAFILE.
- ❏ The DBA section of a Master File cannot have comments within it.

Identifying the DBA: The DBA Attribute

How to:

Change a DBA Password

Example:

Identifying the DBA Using the DBA Attribute

The first security attribute should be a password that identifies the Database Administrator. This password can be up to 64 characters long. It can include special characters. If the DBA password contains blanks, it must be enclosed in single quotation marks. Since nothing else is needed, this line is terminated by the usual delimiter (,$).

Note:

- ❏ Every data source having access limits must have a DBA.
- ❏ Groups of cross-referenced data sources must have the same DBA value.
- ❏ Partitioned data sources, which are read together in the USE command, must have the same DBA value.
- ❏ The Database Administrator has unlimited access to the data source and all cross-referenced data sources. 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 sources without the DBA password.
- ❏ You should thoroughly test every security attribute before the data source 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.

Example: Identifying the DBA Using the DBA Attribute

```
DBA=JONES76,$
```

Procedure: How to Change a DBA Password

The DBA has the freedom to change any of the security attributes. If you change the DBA password in the Master File for an existing FOCUS data source, you must use the RESTRICT command to store the changed DBA password in each FOCUS data source affected by the change. Unless this is done, FOCUS assumes that the new description is an attempt to bypass the restriction rules. You use the following procedure for each data source affected:

1. Edit the Master File, changing the DBA value from old to new.
2. Issue the command:

   ```
   SET PASS=old_DBA_password
   ```

3. Issue the command:

   ```
   RESTRICT
   mastername
   END
   ```

4. Issue the command:

   ```
   SET PASS=new_DBA_password
   ```

Including the DBA Attribute in a HOLD File

With the SET HOLDSTAT command, you can identify a data source containing DBA information and comments to be automatically included in HOLD and PCHOLD Master Files. *Developing Applications* manual.

For z/OS, the data source must be a member in the PDS allocated to ddname MASTER or ERRORS; for CMS, it must have file type MASTER or ERRORS. 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:

```
$================================================================$
$     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 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.

Example: Including a Comment in a HOLD Master File

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 2003/05/20 at 17.58.10 by FOCUS 7.3    $
$            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      ,$
```

Example: Including DBA Attributes in a HOLD Master File

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 2003/05/20 at 17.58.10 by FOCUS 7.3   $
$             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 With Access Rights: The USER Attribute

How to:

Set the USER Attribute

The USER attribute is a password that identifies the users who have legitimate access to the data source. A USER attribute cannot be specified alone; it must be followed by at least one ACCESS restriction (discussed in *Specifying an Access Type: The ACCESS Attribute* on page 385) to specify what sort of ACCESS the user is granted.

Before using a secured data source, a user must enter the password using the SET PASS command. If that password is not included in the Master File, the user is denied access to the data source. When the user does not have a password or has one that is inadequate for the type of access requested, the following message appears:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE:
filename
```

Syntax: How to Set the USER Attribute

Any user whose name or password is not declared in the Master File is denied access to that data source. The syntax of the USER attribute is

```
USER = name
```

where:

name

Is a password of up to 64 characters for the user. The password can include special characters. If the password contains blanks, it must be enclosed in single quotation marks.

For example:

```
USER=TOM,...
```

You can specify a blank password (default value if not previously changed). 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,$
```

Non-Overridable User Passwords (SET PERMPASS)

How to:

Set a Non-Overridable User Password

Reference:

Usage Notes for Non-Overridable User Passwords

Example:

Setting a Non-Overridable User Password

The PERMPASS parameter establishes a user password that remains in effect throughout a session or connection. You can issue this setting in any supported profile but is most useful when established for an individual user by setting it in a user profile. It cannot be set in an ON TABLE phrase. It is recommended that it not be set in FOCPARM or FOCPROF because it would then apply to all users. In a FOCUS session, SET PERMPASS can be issued in PROFILE, a FOCEXEC, or at the command prompt.

All security rules established in the DBA sections of existing Master Files are respected when PERMPASS is in effect. The user cannot issue the SET PASS or SET USER command to change to a user password with different security rules. Any attempt to do so generates the following message:

```
permanent PASS is in effect. Your PASS will not be honored.
VALUE WAS NOT CHANGED
```

Only one permanent password can be established in a session. Once it is set, it cannot be changed within the session.

Syntax: How to Set a Non-Overridable User Password

```
SET PERMPASS=userpass
```

where:

userpass

Is the user password used for all access to data sources with DBA security rules established in their associated Master Files.

Example: Setting a Non-Overridable User Password

Consider the MOVIES Master File with the following DBA rules in effect:

```
DBA=USER1,$
USER = USERR,  ACCESS = R ,$
USER = USERU,  ACCESS = U ,$
USER = USERW,  ACCESS = W ,$
USER = USERRW, ACCESS = RW,$
```

The following FOCEXEC sets a permanent password:

```
SET PERMPASS = USERU
TABLE FILE MOVIES
PRINT TITLE BY DIRECTOR
END
```

The user has ACCESS=U and, therefore, is not allowed to issue a table request against the file:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE:
CAR
BYPASSING TO END OF COMMAND
```

The permanent password cannot be changed:

```
SET PERMPASS = USERRW

permanent PASS is in effect. Your PASS will not be honored.
VALUE WAS NOT CHANGED
```

The user password cannot be changed:

```
SET PASS = USERRW

permanent PASS is in effect. Your PASS will not be honored.
VALUE WAS NOT CHANGED
```

Controlling Case Sensitivity of Passwords

How to:

Control Password Case Sensitivity

Example:

Controlling Password Case Sensitivity

When a DBA or user issues the SET USER, SET PERMPASS or SET PASS command, this user ID is validated before they are given access to any data source whose Master File has DBA attributes. The password is also checked when encrypting or decrypting a FOCEXEC.

The SET DBACSENSITIV command determines whether the password is converted to upper case prior to validation.

Syntax: How to Control Password Case Sensitivity

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

where:

ON

Does not convert passwords to upper case. All comparisons between the password set by the user and the password in the Master File or FOCEXEC are case sensitive.

OFF

Converts passwords to upper case prior to validation. All comparisons between the password set by the user and the password in the Master File or FOCEXEC are *not* case sensitive. OFF is the default value.

Example: Controlling Password Case Sensitivity

Consider the following DBA declaration added to the EMPLOYEE Master File:

```
USER = User2, ACCESS = RW,$
```

User2 wants to report from the EMPLOYEE data source and issues the following command:

```
SET USER = USER2
```

With DBACSENSITIV OFF, User2 can run the request even though the case of the password entered does not match the case of the password in the Master File.

With DBACSENSITIV ON, User2 gets the following message:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE:
```

With DBACSENSITIV ON, the user must issue the following command:

```
SET USER = User2
```

Note: In FOCUS for Mainframe, all user input is transmitted in upper case. Therefore, a mixed case password cannot be issued at the command line. It must be set in a FOCEXEC or profile.

Reference: Usage Notes for Non-Overridable User Passwords

- If you use FOCUSID to set passwords externally, these external passwords may be overridable or non-overridable. If they are non-overridable, they take precedence over the PERMPASS setting.

- SET PERMPASS is supported in MSO. The profiles available in order of precedence are:

  Member PROFILE in the data set allocated to ddname MSOPROF. This profile is executed for all users.

  Members with users' user IDs as names in the data set allocated to ddname MSOPROF. The profile is executed for the user corresponding to the user ID.

  Member PROFILE in the data set allocated by a user to ddname FOCEXEC. This profile is executed for that specific user and is a standard FOCUS profile.

  Member SHELPROF in the data set allocated to ddname FOCEXEC in the MSO startup JCL. This profile is executed for all users and is a standard FOCUS profile.

Establishing User Identity

How to:

Establish User Identity

Example:

Establishing User Identity

A user must enter his or her password before using any FOCUS data source 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.

Syntax: How to Establish User Identity

```
SET {PASS|USER} = name [[IN {file|* [NOCLEAR]}], name [IN file] ...]
```

where:

name

Is the user's name or password. If a character used in the password has a special meaning in your operating environment (for example, as an escape character), you can issue the SET USER command in a FOCEXEC and execute the FOCEXEC to set the password. If the password contains a blank, you do not have to enclose it in single quotation marks when issuing the SET USER command.

file

Is the name of the Master File to which the password applies.

*

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.

Example: Establishing User Identity

In the following example, the password TOM is in effect for all data sources 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
```

A list of the files for which a user has set specific passwords is maintained. 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 data sources secured with different passwords.

Users must issue passwords using the SET PASS command during each session in which they use a secured data source. They may issue passwords at any time before using the data source and can issue a different password afterward to access another data source.

Specifying an Access Type: The ACCESS Attribute

In this section:

The ACCESS attribute specifies what sort of access a user is granted. Every security declaration, except the DBA declaration, 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 data source.

You can assign the ACCESS attribute one of four values. These are:

`ACCESS=R`	Read-only
`ACCESS=W`	Write only
`ACCESS=RW`	Read the data source and write new segment instances
`ACCESS=U`	Update only

Access levels affect what kind of commands a user can issue. Before you decide what access levels to assign to a user, consider what commands that user will need. If a user does not have sufficient access rights to use a given command, the following message appears:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE:
filename
```

ACCESS levels determine what a user can do to the data source. Use the RESTRICT attribute (discussed in *Limiting Data Source Access: The RESTRICT Attribute* on page 390) 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 data source.

Types of Access

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 granted.

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
MAINTAIN		X	X	X	X
MATCH	X		X		X
MODIFY		X	X	X	X
REBUILD			X		X
RESTRICT					X
SCAN			X	X	X
TABLE	X		X		X

Reference: 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 Data Source Access: The RESTRICT Attribute* on page 390 for more information on the RESTRICT attribute.

The RESTRICT attribute keywords affect the resulting HOLD file created by the CHECK command 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 the user can reference these.

`VALUE`
: Fields named in the VALUE parameter are included since the user can reference these.

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.

CREATE Command

Only users with the DBA password or read/write (RW) access rights can issue a CREATE command.

DECRYPT Command

Only users with the DBA password can issue a DECRYPT command.

Any attempt to decrypt a Master File that contains DBA when MASTER=OLD generates the following message:

```
(FOC209) THE DATA VALUE EXCEEDS ITS LENGTH SPECIFICATION
```

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 data source 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).

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 Data Source Access: The RESTRICT Attribute* on page 390 for information on these restrictions.)

The password is placed in the File Control Block (FCB), words 19 and 20 (byte 73 to 80).

MODIFY or MAINTAIN Command

Users with ACCESS=W, RW, or U can use the MODIFY or MAINTAIN command. In MODIFY or MAINTAIN, 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 or MAINTAIN features. New instances of data may be added to a data source and old ones deleted; existing values may be updated.

Users with ACCESS=R (read only) can use the MAINTAIN command to retrieve information from the data source. Users with read-only access cannot use the MODIFY command.

REBUILD Command

Only users with the DBA password or read/write (RW) access rights can issue the REBUILD command. This command is only for FOCUS data sources.

RESTRICT Command

Only users with the DBA password may use the RESTRICT command.

FSCAN Facility

Users with ACCESS=RW have unlimited access to the data source, except for any restrictions imposed by the RESTRICT or NAME attributes. Users with ACCESS=U can display the entire data source, 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 data source 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 data source. 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.

SCAN Facility

The rules for accessing a data source are the same as for FSCAN except that, in addition, users with ACCESS=W cannot use SCAN.

TABLE or MATCH 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 Data Source Access: The RESTRICT Attribute

In this section:

How to:

Example:

The ACCESS attribute determines what a user can do with a data source. The optional RESTRICT attribute further restricts a user's access to certain fields, values, or segments.

Syntax: How to Limit Data Source Access

```
...RESTRICT=level, NAME={name|SYSTEM} [,VALUE=test],$
```

where:

level

Can be one of the following:

FIELD specifies that the user cannot access the fields named with the NAME parameter.

SEGMENT specifies that the user cannot access the segments named with the NAME parameter.

PROGRAM specifies that the program named with the NAME parameter will be called whenever the user uses the data source (discussed in *Program Accounting/Resource Limitation* on page 411).

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 appear. This option is not supported with relational data sources.

name

Is the name of the field or segment 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 data source, 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.

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.

Example: Limiting Data Source Access

```
USER=BILL ,ACCESS=R ,RESTRICT=SEGMENT ,NAME=COMPSEG,$
```

Restricting Access to a Field or a Segment

How to:

Restrict Access to a Field or a Segment

Example:

Restricting Access to a Segment

Reusing a Common Set of Access Restrictions

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 data source. Users may be limited to reading, writing, or updating new records, but every record in the data source is available for the operation.

Syntax: How to Restrict Access to a Field or a Segment

```
...RESTRICT=level,  NAME=name,$
```

where:

level

Can be one of the following:

FIELD specifies that the user cannot access the fields named with the NAME parameter.

SEGMENT specifies that the user cannot access the segments named with the NAME parameter.

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 appear. When used after NOPRINT, NAME can only be a field name. This option is not supported with relational data sources.

name

Is the name of the field or segment 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 data source, including descendant segments. Multiple fields or segments can be specified by issuing the RESTRICT attribute several times for one user.

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 appear. 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, 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.

Example: Restricting Access to a Segment

In the following example, Bill has read-only access to everything in the data source except the COMPSEG segment:

```
USER=BILL ,ACCESS=R ,RESTRICT=SEGMENT ,NAME=COMPSEG,$
```

Example: Reusing a Common Set of Access Restrictions

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,$
```

Note: A restriction on a segment also affects access to its descendants.

Restricting Access to a Value

How to:

Restrict Values a User Can Read

Example:

Restricting Values a User Can Read

You can also restrict the values to which a user has access by providing a test condition in your RESTRICT attribute. The user is restricted to using only those values that satisfy the test condition.

You can restrict values in one of two ways: by restricting the values the user can read from the data source, or restricting what the user can write to a data source. 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 attribute. A RESTRICT attribute 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.

Note: RESTRICT=VALUE is not supported in Maintain.

You restrict the values a user can write to a data source by setting ACCESS=W and RESTRICT=VALUE. This type of restriction, which functions as a VALIDATE command 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 command in MODIFY. You can also use ACCESS=W and RESTRICT=VALUE to limit the data values in the data source for which a user can provide new values. When ACCESS=W, the user will be able to access all data values in the data source. 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 attributes: 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 407. 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.

Syntax: How to Restrict Values a User Can Read

```
...ACCESS=R, RESTRICT=VALUE, NAME=name, VALUE=test,$
```

where:

name

Is the name of the segment which, if referenced, activates the test. To specify all segments in the data source, specify NAME=SYSTEM.

test

Is the test being performed.

Example: Restricting Values a User Can Read

```
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 attributes 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, 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 data source could be retrieved, but with the salary information of only those meeting the test condition.

Restricting Values a User Can Write

How to:

Restrict Values a User Can Write

Restrict Values a User Can Enter in a Segment

Example:

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 Databases* manual.

There are two different ways you can restrict the values a user can write to a data source: by restricting the values the user actually is allowed to enter (global validate), or by restricting the values that the user is allowed to change (ON MATCH VALIDATE). You must supply an ACCESS=R restriction to restrict the user from seeing certain data values in the data source.

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.

Syntax: How to Restrict Values a User Can Write

```
...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 source values, therefore, 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.

Syntax: How to Restrict Values a User Can Enter in a Segment

If your MODIFY procedure contains MATCH commands, 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.

```
...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.

Example: Restricting Values a User Can Write

This example 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.

```
(A) USER=CHUCK ,ACCESS=W ,RESTRICT=VALUE ,
        NAME=CHRANGE,
        VALUE=SALARY LT 20000 AND SALARY GT 5000,$

(B) USER=CHUCK ,ACCESS=W ,RESTRICT=VALUE ,NAME=COMPSEG,
        VALUE=SALARY LT 20000 AND SALARY GT 5000,$
```

The difference between the restriction created in 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 for COMPSEG right before the action (UPDATE or DELETE) and can reference D. fields.

Restricting Values a User Can Alter

How to:

Restrict Values a User Can Alter

Example:

Restricting Values a User Can Alter

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 data source 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.

Syntax: How to Restrict Values a User Can Alter

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.

Example: Restricting Values a User Can Alter

```
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 source 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 acceptable 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 proves useful to issue both ACCESS=W (for MODIFY) and ACCESS=R (for TABLE) value restrictions for a user. This limits the values a user can write to the data source and limits the data values that the user can actually see. Do this by issuing a RESTRICT=VALUE attribute with ACCESS=R to prohibit the user from seeing any values other than those specified in the test condition. You then issue a RESTRICT=VALUE attribute with ACCESS=W that specifies the write restrictions placed on the user. You cannot use ACCESS=RW to do this.

Example: Restricting Both Read and Write Values for a User

```
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 Master File

In this section:

File Naming Requirements for DBAFILE

Connection to an Existing DBA System With DBAFILE

Combining Applications With DBAFILE

How to:

Place Security Attributes in a Central Master File

Example:

Placing Security Attributes in a Central Master File

DBAFILE Naming Conventions

Connecting to an Existing DBA System With DBAFILE

The DBAFILE attribute enables you to place all passwords and restrictions for multiple 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, including:

- Passwords only have to be stored once when they are applicable to a group of data sources, simplifying password administration.
- Data sources with different user passwords can be JOINed or COMBINEd. In addition, individual DBA information remains in effect for each data source 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.

Note:

- All Master Files that specify the same DBAFILE have the same DBA password.
- The central DBAFILE, like any Master File, must contain at least one segment declaration and one field declaration before the END statement that signifies the presence of DBA information. Even if a required attribute is not assigned a specific value, it must be represented by a comma. 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 Master Files should be encrypted.
- The DBAFILE may contain a list of passwords and restrictions following the DBA password. These passwords apply to all data sources that reference this DBAFILE. In the example in *Placing Security Attributes in a Central Master File* on page 401, PASS=BILL, with ACCESS=R (read only), applies to all data sources that contain the attribute DBAFILE=FOUR.
- After the common passwords, the DBAFILE may specify data source-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). See *File Naming Requirements for DBAFILE* on page 402 for additional information about the FILENAME attribute.
- Data source-specific restrictions override general restrictions for the specified data source. 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 data source.
- 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 data sources, but in FILENAME=THREE it carries an extra restriction, RESTRICT=..., which applies only to data source THREE.

Syntax: How to Place Security Attributes in a Central Master File

```
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.

Example: Placing Security Attributes in a Central Master File

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,$
```

File Naming Requirements for DBAFILE

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 unknown by the DBAFILE.

Example: DBAFILE Naming Conventions

```
ONE MASTER
FILENAME=XONE
   .
   .
   .
END
DBA=ABC, DBAFILE=FOUR,$


FOUR MASTER
FILENAME=XFOUR
   .
   .
   .
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.

For security reasons, the FILENAME attribute in the Master File containing the DBAFILE information should *not* be the same as the name of that Master File. Note that in Master File FOUR, the FILENAME attribute specifies the name XFOUR.

Connection to an Existing DBA System With DBAFILE

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 data sources is JOINed, the first data source in the list is the controlling data source. Its passwords are the only ones examined. For a COMBINE, only the last data source's passwords take effect. All data sources must have the same DBA password.

In the new system, the DBA sections of all data sources 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 data source active in a JOIN list or COMBINE, specify DBAFILE for that data source.

After you start to use the new system, 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 data source itself.

Example: Connecting to an Existing DBA System With DBAFILE

```
FILENAME=SEVEN,
  SEGNAME=..
    FIELDNAME=...
      .
      .
      .
END
DBA=ABC,DBAFILE=SEVEN,$     (OR DBAFILE= ,$)
 PASS=...
 PASS=...
```

Combining Applications With DBAFILE

Since each data source now contributes its own restrictions, you can JOIN and COMBINE data sources that come from different applications and have different user passwords. The only requirement is a valid password for each data source. 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.

You can assign screening conditions to a data source that are automatically applied to any report request that accesses the data source. 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 data source.
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: R read-only W write new segments only RW read and write U update values only
RESTRICT	RESTRICT	8	Types of restrictions to be imposed for this access level. Values are: SEGMENT FIELD VALUE SAME PROGRAM NOPRINT
NAME	NAME	66	Name of segment or field restricted or 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 containing passwords and restrictions to use.

Hiding Restriction Rules: The ENCRYPT Command

In this section:

How to:

Example:

Since the restriction information for a FOCUS data source is stored in its Master File, encrypt the Master File in order to prevent users from examining the restriction rules. Only the Database Administrator can encrypt a description. You must set PASS=DBA*name* 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.

Syntax: How to Hide Restriction Rules: ENCRYPT Command

```
ENCRYPT FILE filename
```

where:

filename

Is the name of the file to be encrypted.

Example: Encrypting and Decrypting a Master File

The following is an example of the complete procedure:

```
SET PASS=JONES76
ENCRYPT FILE PERS
```

The process can be reversed in order 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 parameter within the Master File to encrypt some or all of its segments. When encrypted files are stored on the external media (disk or tape) each is 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.

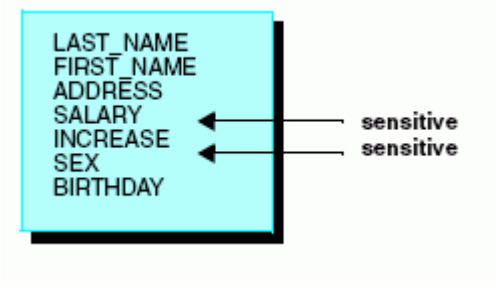
Example: Encrypting Data

```
SEGMENT=COMPSEG, PARENT=IDSEG, SEGTYPE=S1, ENCRYPT=ON,$
```

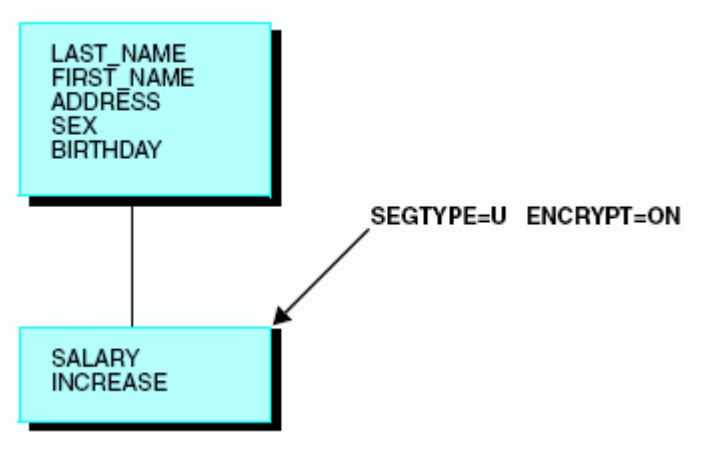
You must specify the ENCRYPT parameter before entering any data in the data source. The message NEW FILE... must appear when the encryption is first requested. Encryption cannot be requested later by a change to the Master File and cannot be removed after it has been requested or any data has been entered in the data source.

Performance Considerations for Encrypted Data

There is a small loss in processing efficiency when data is encrypted. 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 the original segment. For example, suppose the data items on a segment are:



They should be grouped as:



Note: If you change the DBA password, you must issue the RESTRICT command, as described in *How to Change a DBA Password* on page 376.

Displaying the Decision Table

When entering security attributes for a Master File, FOCUS creates an internal decision table that lists users and the access privileges. You can display the decision table associated with a given data source whenever the Master File of the data source 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.

Syntax: How to Display the Decision Table

```
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

Displays 18 characters in the NAME column.

Note: The DBATABLE procedure is supplied with FOCUS. Contact your system administrator if you cannot locate it.

Example: Displaying the Decision Table

The following example displays the decision table for the pers data source.

```
>
ex dbatable
FILE NAME:
pers
>
>


 PAGE     1

 DATA BASE ADMINISTRATOR JONES76
 DBAFILE PERS      ON 05/20/03 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'
```

Setting a Password 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 first enters 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. The passwords 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 data sources being accessed.

See your installation documentation for FOCUSID installation instructions.

FOCEXEC Security

In this section:

Encrypting and Decrypting a FOCEXEC

Most data security issues are best handled by the FOCUS DBA facility. However, 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 with -CRTFORM to create a password prompt with no display of the input characters.

Syntax: How to Suppress Password Display

```
<.NODISP.&mypass
```

Example: Suppressing Password Display

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 a Password in an Encrypted FOCEXEC

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 command, 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 printing the procedure cannot reveal the password.

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 editing the FOCEXEC. It does not appear when the ECHO option is ALL and is not printed in a batch run log.

Encrypting and Decrypting a FOCEXEC

Keep the actual text of a stored FOCEXEC confidential while allowing users to execute the FOCEXEC. You 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 encrypting password can decrypt the FOCEXEC.

The password selected by a user to ENCRYPT or DECRYPT a FOCEXEC is not viewable by any editor and it is unrelated to the DBA passwords of the files being used.

Syntax: How to Encrypt and Decrypt a FOCEXEC

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 or have the commands displayed on the terminal, so &ECHO has no effect.

Locking a FOCEXEC User Out of FOCUS

Users can 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 Dialogue Manager produces an error message. Following the error message, the user is 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 this section:

In addition to controlling access to a data source, 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 use of the data source. 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 data source security, as well as to external security systems.

Program Accounting

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. Activate this user-written program by assigning the value PROGRAM to the RESTRICT attribute. The program is passed the statistics of the run, or 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 data sources. For example, the 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 data source.
- 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

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 VALUE specified for this user (8 bytes).

For example:

```
USER=PETER, ACCESS=R, RESTRICT=PROGRAM,
NAME=PETER1, VALUE=D76,$
```

Calls the program Peter1 which will return a value. If the value is D76, then this passes DBA.

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 that 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 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 data source.
NUMB	Int	4 bytes	The number of data and defined fields in the data source.
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 limits the number of records retrieved only if fields from segment COMPSEG are referred to in the report request.

For non-FOCUS data sources, READLIMIT EQ can be used exactly as RECORDLIMIT EQ to set an automatic maximum on the number of successful reads issued for sequential data sources 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 your installation documentation.

Absolute File Integrity

How to:

Invoke Absolute File Integrity

Invoke Absolute File Integrity for an Existing Data Source

FOCUS can perform shadow paging to guarantee the integrity of any FOCUS data source 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 data source.

FOCUS shadow paging is accomplished by checkpoints and directory pages, which, in one stroke, changes the shadow pages into current database pages. Basically, FOCUS creates a shadow image of a FOCUS data source, with each FOCUS page having a corresponding shadow page. At any point, one of the data source images has complete data integrity, regardless of what happens to the other. Therefore, the data integrity of a FOCUS data source will never be compromised by a system crash or other circumstances.

Absolute File Integrity is available for FOCUS data sources 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 data source correctly but IBM does not guarantee integrity on these data sources.
- Absolute file integrity is not supported for XFOCUS data sources.

Syntax: How to Invoke Absolute File Integrity

To invoke Absolute File Integrity in FOCUS before creating the data source with the CREATE FILE command, issue the following

```
SET SHADOW = value
```

where:

value

Can be one of the following:

OFF does not invoke Absolute File Integrity. OFF 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 data sources was increased), meaning fewer pages are shadowed. If your FOCUS data source was created with this option, the maximum number of pages is 63,551. If this limit is exceeded, FOCUS displays the (FOC198) error message.

Procedure: How to Invoke Absolute File Integrity for an Existing Data Source

If the data source 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 data source with the CREATE FILE command.
4. Specify the REBUILD, REORG, and LOAD options with the REBUILD command.

A Master Files and Diagrams

This appendix contains descriptions and structure diagrams for the sample data sources used throughout the documentation.

Topics:

Creating Sample Data Sources

Create sample data sources on your user ID by executing the procedures specified below. These FOCEXECs are supplied 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.

Data Source	Load Procedure Name

EMPLOYEE, EDUCFILE, and JOBFILE	Under CMS, enter:
	`EX EMPTEST`
	Under z/OS, enter:
	`EX EMPTSO`
	These FOCEXECs also test the data sources 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 data sources already exist on your user ID, the FOCEXEC replaces them with new copies. This FOCEXEC assumes that the high-level qualifier for the FOCUS data sources is the same as the high-level qualifier for the MASTER PDS that was unloaded from the tape.

SALES PROD	`EX SALES` `EX PROD`
CAR	`EX CARLOAD` (`CARTEST` creates it automatically during installation).

LEDGER
FINANCE
REGION
COURSES
EXPERSON
	`EX LEDGER`
	`EX FINANCE`
	`EX REGION`
	`EX COURSES`
	`EX EXPERSON`

Data Source	Load Procedure Name
EMPDATA TRAINING COURSE JOBHIST JOBLIST LOCATOR PERSINFO SALHIST	`EX LOADPERS`
PAYHIST	None (PAYHIST DATA is a sequential data source and is allocated during the installation process).
COMASTER	None (COMASTER is used for debugging other Master Files).
VIDEOTRK, MOVIES, and ITEMS	`EX LOADVTRK`
VIDEOTR2	`EX LOADVID2`
Gotham Grinds	`EX DBLGG`

Century Corp:
CENTCOMP
CENTFIN
CENTHR
CENTINV
CENTORD
CENTQA
CENTGL
CENTSYSF
CENTSTMT
	
	`EX LOADCOM`
	`EX LOADFIN`
	`EX LOADHR`
	`EX LOADINV`
	`EX LOADORD`
	`EX LOADCQA`
	`EX LDCENTGL`
	`EX LDCENTSY`
	`EX LDSTMT`

EMPLOYEE Data Source

In this section:

EMPLOYEE contains sample data about a company's employees. Its segments are:

`EMPINFO`
: Contains employee IDs, names, and positions.

`FUNDTRAN`
: Specifies employees' direct deposit accounts. This segment is unique.

`PAYINFO`
: Contains the employees' salary history.

`ADDRESS`
: Contains employees' home and bank addresses.

`SALINFO`
: Contains data on employees' monthly pay.

`DEDUCT`
: Contains data on monthly pay deductions.

EMPLOYEE also contains cross-referenced segments belonging to the JOBFILE and EDUCFILE files, also described in this appendix. The segments are:

`JOBSEG (from JOBFILE)`
: Describes the job positions held by each employee.

`SKILLSEG (from JOBFILE)`
: Lists the skills required by each position.

`SECSEG (from JOBFILE)`
: Specifies the security clearance needed for each job position.

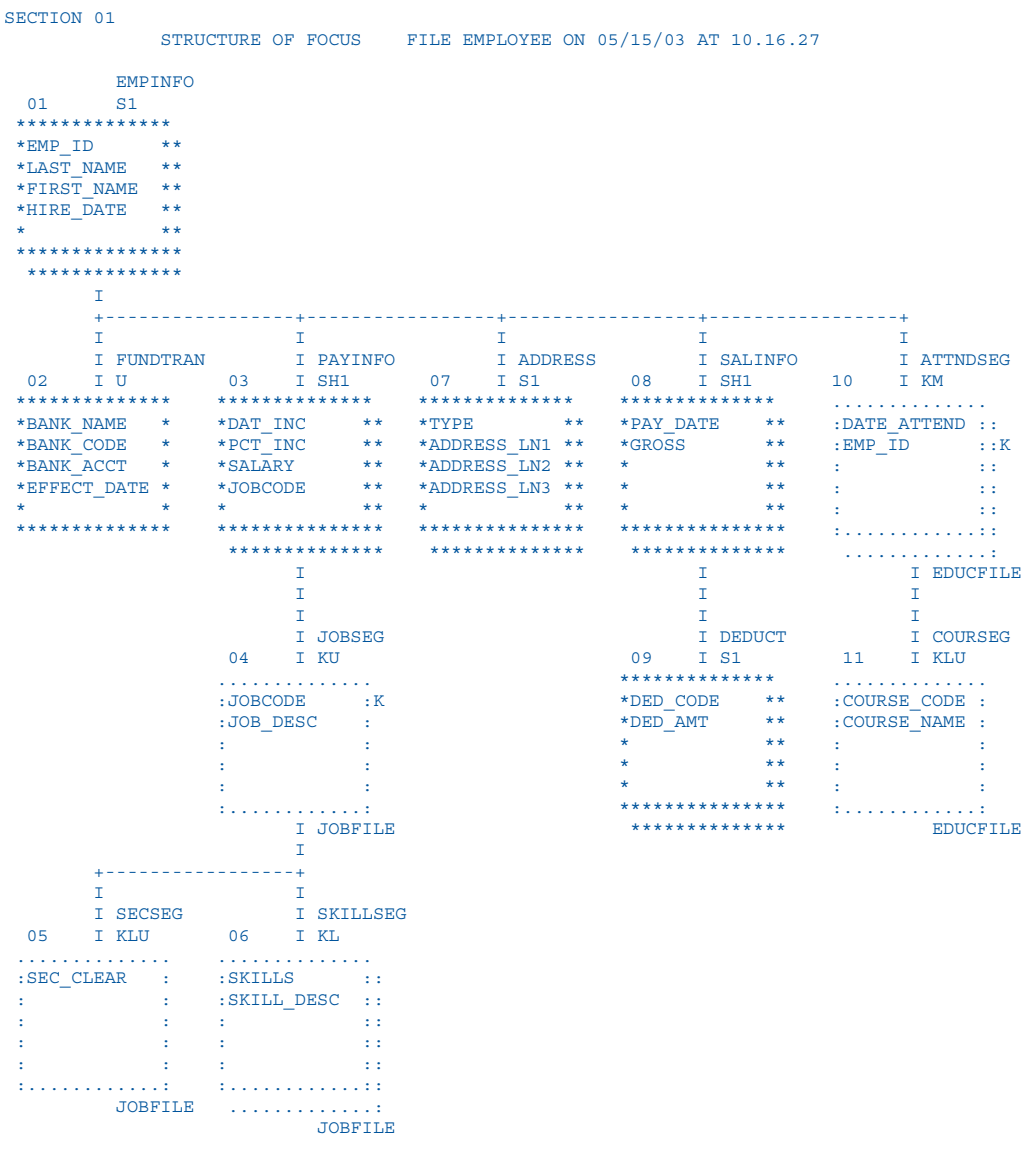
`ATTNDSEG (from EDUCFILE)`
: Lists the dates that employees attended in-house courses.

`COURSEG (from EDUCFILE)`
: Lists the courses that the employees attended.

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,$
```

EMPLOYEE Structure Diagram



JOBFILE Data Source

In this section:

JOBFILE contains sample data about 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.

JOBFILE Master File

```
FILENAME=JOBFILE,  SUFFIX=FOC
 SEGNAME=JOBSEG,    SEGTYPE=S1
  FIELDNAME=JOBCODE,     ALIAS=JC, FORMAT=A3,     INDEX=I,$
  FIELDNAME=JOB_DESC,    ALIAS=JD, FORMAT=A25            ,$
 SEGNAME=SKILLSEG, SEGTYPE=S1,   PARENT=JOBSEG
  FIELDNAME=SKILLS,      ALIAS=,    FORMAT=A4            ,$
  FIELDNAME=SKILL_DESC, ALIAS=SD, FORMAT=A30           ,$
 SEGNAME=SECSEG,    SEGTYPE=U,    PARENT=JOBSEG
  FIELDNAME=SEC_CLEAR,  ALIAS=SC, FORMAT=A6            ,$
```

JOBFILE Structure Diagram

```
SECTION 01

              STRUCTURE OF FOCUS    FILE JOBFILE ON 05/15/03 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 **
 *            *    *           **
 *            *    *           **
 *            *    *           **
 **************    **************
                    *************
```

EDUCFILE Data Source

In this section:

EDUCFILE contains sample data about 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.

EDUCFILE Master File

```
FILENAME=EDUCFILE, SUFFIX=FOC
 SEGNAME=COURSEG,  SEGTYPE=S1
  FIELDNAME=COURSE_CODE,  ALIAS=CC,   FORMAT=A6,           $
  FIELDNAME=COURSE_NAME,  ALIAS=CD,   FORMAT=A30,          $
 SEGNAME=ATTNDSEG,  SEGTYPE=SH2,  PARENT=COURSEG
  FIELDNAME=DATE_ATTEND,  ALIAS=DA,   FORMAT=I6YMD,        $
  FIELDNAME=EMP_ID,       ALIAS=EID,  FORMAT=A9, INDEX=I,  $
```

EDUCFILE Structure Diagram

```
SECTION 01

              STRUCTURE OF FOCUS    FILE EDUCFILE ON 05/15/03 AT 14.45.44


          COURSEG
  01      S1
 **************
 *COURSE_CODE **
 *COURSE_NAME **
 *            **
 *            **
 *            **
 ***************
  **************
         I
         I
         I
         I ATTNDSEG
  02     I SH2
 **************
 *DATE_ATTEND **
 *EMP_ID      **I
 *            **
 *            **
 *            **
 ***************
  **************
```

SALES Data Source

In this section:

SALES contains sample data about a dairy company with an affiliated 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. The PROD_CODE field is indexed. The RETURNS and DAMAGED fields have the MISSING=ON attribute.

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,$
```

SALES Structure Diagram

```
SECTION 01
              STRUCTURE OF FOCUS    FILE SALES ON 05/15/03 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 **
 *            **
 ***************
  **************
```

PROD Data Source

The PROD data source lists products sold by a dairy company. It consists of one segment, PRODUCT. The field PROD_CODE is indexed.

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, $
```

PROD Structure Diagram

```
SECTION 01
             STRUCTURE OF FOCUS   FILE PROD   ON 05/15/03 AT 14.57.38
        PRODUCT
01      S1
**************
*PROD_CODE    **I
*PROD_NAME    **
*PACKAGE      **
*UNIT_COST    **
*             **
***************
 **************
```

CAR Data Source

In this section:

CAR contains sample data about 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.

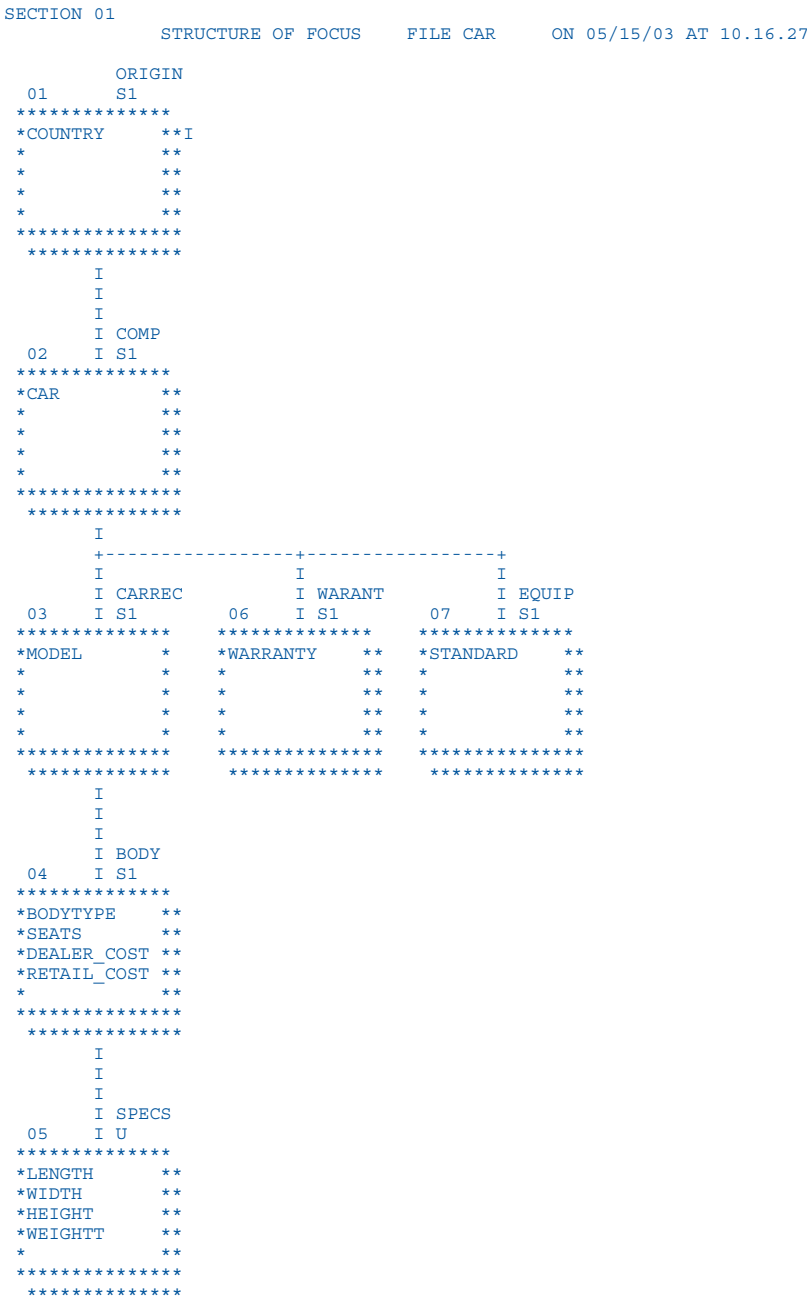
EQUIP
: Lists standard equipment.

The aliases in the CAR Master File are specified without the ALIAS keyword.

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,$
```

CAR Structure Diagram



LEDGER Data Source

In this section:

LEDGER contains sample accounting data. It consists of one segment, TOP. This data source is specified primarily for FML examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

LEDGER Master File

```
FILENAME=LEDGER, SUFFIX=FOC,$
 SEGNAME=TOP,    SEGTYPE=S2,$
  FIELDNAME=YEAR    ,   , FORMAT=A4, $
  FIELDNAME=ACCOUNT,   , FORMAT=A4, $
  FIELDNAME=AMOUNT ,   , FORMAT=I5C,$
```

LEDGER Structure Diagram

```
SECTION 01

            STRUCTURE OF FOCUS    FILE LEDGER   ON 05/15/03 AT 15.17.08

          TOP
  01      S2
 **************
 *YEAR         **
 *ACCOUNT      **
 *AMOUNT       **
 *             **
 *             **
 ***************
  **************
```

FINANCE Data Source

In this section:

FINANCE contains sample financial data for balance sheets. It consists of one segment, TOP. This data source is specified primarily for FML examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

FINANCE Master File

```
FILENAME=FINANCE, SUFFIX=FOC,$
 SEGNAME=TOP,     SEGTYPE=S2,$
  FIELDNAME=YEAR    ,   , FORMAT=A4,   $
  FIELDNAME=ACCOUNT,   , FORMAT=A4,   $
  FIELDNAME=AMOUNT ,   , FORMAT=D12C,$
```

FINANCE Structure Diagram

```
SECTION 01

            STRUCTURE OF FOCUS    FILE FINANCE  ON 05/15/03 AT 15.17.08

          TOP
  01      S2
 **************
 *YEAR         **
 *ACCOUNT      **
 *AMOUNT       **
 *             **
 *             **
 ***************
  **************
```

REGION Data Source

In this section:

REGION contains sample account data for the eastern and western regions of the country. It consists of one segment, TOP. This data source is specified primarily for FML examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

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,$
```

REGION Structure Diagram

```
SECTION 01

            STRUCTURE OF FOCUS    FILE REGION   ON 05/15/03 AT 15.18.48

          TOP
  01      S1
 **************
 *ACCOUNT     **
 *E_ACTUAL    **
 *E_BUDGET    **
 *W_ACTUAL    **
 *            **
 ***************
  **************
```

COURSES Data Source

COURSES contains sample data about education courses. It consists of one segment, CRSESEG1. The field DESCRIPTION has a format of TEXT (TX).

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,              $
```

COURSES Structure Diagram

```
SECTION 01

             STRUCTURE OF FOCUS    FILE COURSES   ON 05/15/03 AT 12.26.05

         CRSESEG1
 01      S1
**************
*COURSE_CODE **I
*COURSE_NAME **
*DURATION    **
*DESCRIPTION **T
*            **
***************
 **************
```

EXPERSON Data Source

In this section:

EXPERSON Master File

The EXPERSON data source contains personal data about individual employees. It consists of one segment, ONESEG. 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       ,$
```

EXPERSON Structure Diagram

```
SECTION 01

            STRUCTURE OF FOCUS    FILE EXPERSON  ON 05/15/03 AT 14.50.58

          ONESEG
  01      S1
 **************
 *SOC_SEC_NO  **
 *FIRST_NAME  **
 *LAST_NAME   **
 *AGE         **
 *            **
 ***************
  **************
```

EMPDATA Data Source

In this section:

EMPDATA contains sample data about a company's employees. It consists of one segment, EMPDATA. The PIN field is indexed. The AREA field is a temporary field.

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');$
```

EMPDATA Structure Diagram

```
SECTION 01

            STRUCTURE OF FOCUS    FILE EMPDATA  ON 05/15/03 AT 14.49.09

          EMPDATA
  01      S1
 **************
 *PIN          **I
 *LASTNAME     **
 *FIRSTNAME    **
 *MIDINITIAL   **
 *             **
 ***************
  **************
```

TRAINING Data Source

In this section:

TRAINING Master File

TRAINING Structure Diagram

TRAINING contains sample data about training courses for employees. It consists of one segment, TRAINING. The PIN field is indexed. The EXPENSES, GRADE, and LOCATION fields have the MISSING=ON attribute.

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, $
```

TRAINING Structure Diagram

```
SECTION 01

            STRUCTURE OF FOCUS    FILE TRAINING ON 05/15/03 AT 14.51.28

          TRAINING
  01      SH3
 **************
 *PIN          **I
 *COURSESTART **
 *COURSECODE  **
 *EXPENSES    **
 *            **
 ***************
  **************
```

COURSE Data Source

In this section:

COURSE contains sample data about education courses. It consists of one segment, CRSELIST.

COURSE Master File

```
FILENAME=COURSE,    SUFFIX=FOC
 SEGNAME=CRSELIST,  SEGTYPE=S1
  FIELDNAME=COURSECODE,  ALIAS=CCOD,   FORMAT=A7,    INDEX=I,     $
  FIELDNAME=CTITLE,      ALIAS=COURSE, FORMAT=A35,                 $
  FIELDNAME=SOURCE,      ALIAS=ORG,    FORMAT=A35,                 $
  FIELDNAME=CLASSIF,     ALIAS=CLASS,  FORMAT=A10,                 $
  FIELDNAME=TUITION,     ALIAS=FEE,    FORMAT=D8.2, MISSING=ON,    $
  FIELDNAME=DURATION,    ALIAS=DAYS,   FORMAT=A3,   MISSING=ON,    $
  FIELDNAME=DESCRIPTN1,  ALIAS=DESC1,  FORMAT=A40,                 $
  FIELDNAME=DESCRIPTN2,  ALIAS=DESC2,  FORMAT=A40,                 $
  FIELDNAME=DESCRIPTN2,  ALIAS=DESC3,  FORMAT=A40,                 $
```

COURSE Structure Diagram

```
SECTION 01

             STRUCTURE OF FOCUS    FILE COURSE   ON 05/15/03 AT 12.26.05

         CRSELIST
 01      S1
**************
*COURSECODE  **I
*CTITLE      **
*SOURCE      **
*CLASSIF     **
*            **
***************
 **************
```

JOBHIST Data Source

In this section:

JOBHIST contains information about an employee's jobs. Both the PIN and JOBSTART fields are keys. The PIN field is indexed.

JOBHIST Master File

```
FILENAME=JOBHIST, SUFFIX=FOC
SEGNAME=JOBHIST, SEGTYPE=SH2
 FIELDNAME=PIN,          ALIAS=ID,       FORMAT=A9,      INDEX=I ,$
 FIELDNAME=JOBSTART,     ALIAS=SDAT,     FORMAT=YMD,               $
 FIELDNAME=JOBCLASS,     ALIAS=JCLASS,   FORMAT=A8,                $
 FIELDNAME=FUNCTITLE,    ALIAS=FUNC,     FORMAT=A20,               $
```

JOBHIST Structure Diagram

```
SECTION 01
              STRUCTURE OF FOCUS    FILE JOBHIST  ON 01/22/08 AT 16.23.46

          JOBHIST
  01      SH2
 **************
 *PIN          **I
 *JOBSTART     **
 *JOBCLASS     **
 *FUNCTITLE    **
 *             **
 ***************
  **************
```

JOBLIST Data Source

In this section:

JOBHIST contains information about jobs. The JOBCLASS field is indexed.

JOBLIST MASTER File

```
FILENAME=JOBLIST, SUFFIX=FOC
SEGNAME=JOBSEG,   SEGTYPE=S1
 FIELDNAME=JOBCLASS,     ALIAS=JCLASS,   FORMAT=A8,     INDEX=I ,$
 FIELDNAME=CATEGORY,     ALIAS=JGROUP,   FORMAT=A25,             $
 FIELDNAME=JOBDESC,      ALIAS=JDESC,    FORMAT=A40,             $
 FIELDNAME=LOWSAL,       ALIAS=LSAL,     FORMAT=D12.2M,          $
 FIELDNAME=HIGHSAL,      ALIAS=HSAL,     FORMAT=D12.2M,          $

DEFINE GRADE/A2=EDIT  (JCLASS,'$$$99');$
DEFINE LEVEL/A25=DECODE GRADE (08 'GRADE 8' 09 'GRADE 9' 10
'GRADE 10' 11 'GRADE 11' 12 'GRADE 12' 13 'GRADE 13' 14 'GRADE 14');$
```

JOBLIST Structure Diagram

```
SECTION 01
              STRUCTURE OF FOCUS    FILE JOBLIST  ON 01/22/08 AT 16.24.52

          JOBSEG
  01      S1
 **************
 *JOBCLASS     **I
 *CATEGORY     **
 *JOBDESC      **
 *LOWSAL       **
 *             **
 ***************
  **************
```

LOCATOR Data Source

In this section:

JOBHIST contains information about an employee's location and phone number. The PIN field is indexed.

LOCATOR MASTER File

```
FILENAME=LOCATOR, SUFFIX=FOC
SEGNAME=LOCATOR,   SEGTYPE=S1,
 FIELDNAME=PIN,           ALIAS=ID_NO,     FORMAT=A9,    INDEX=I, $
 FIELDNAME=SITE,          ALIAS=SITE,      FORMAT=A25,            $
 FIELDNAME=FLOOR,         ALIAS=FL,        FORMAT=A3,             $
 FIELDNAME=ZONE,          ALIAS=ZONE,      FORMAT=A2,             $
 FIELDNAME=BUS_PHONE,     ALIAS=BTEL,      FORMAT=A5,             $
```

LOCATOR Structure Diagram

```
SECTION 01
              STRUCTURE OF FOCUS    FILE LOCATOR  ON 01/22/08 AT 16.26.55

          LOCATOR
  01      S1
 **************
 *PIN          **I
 *SITE         **
 *FLOOR        **
 *ZONE         **
 *             **
 ***************
  **************
```

PERSINFO Data Source

In this section:

PERSINFO contains an employee's personal information. The PIN field is indexed.

PERSINFO MASTER File

```
FILENAME=PERSINFO, SUFFIX=FOC
SEGNAME=PERSONAL, SEGTYPE=S1
 FIELDNAME=PIN,           ALIAS=ID,       FORMAT=A9,    INDEX=I,     $
 FIELDNAME=INCAREOF,      ALIAS=ICO,      FORMAT=A35,                $
 FIELDNAME=STREETNO,      ALIAS=STR,      FORMAT=A20,                $
 FIELDNAME=APT,           ALIAS=APT,      FORMAT=A4,                 $
 FIELDNAME=CITY,          ALIAS=CITY,     FORMAT=A20,                $
 FIELDNAME=STATE,         ALIAS=PROV,     FORMAT=A4,                 $
 FIELDNAME=POSTALCODE,    ALIAS=ZIP,      FORMAT=A10,                $
 FIELDNAME=COUNTRY,       ALIAS=CTRY,     FORMAT=A15,                $
 FIELDNAME=HOMEPHONE,     ALIAS=TEL,      FORMAT=A10,                $
 FIELDNAME=EMERGENCYNO,   ALIAS=ENO,      FORMAT=A10,                $
 FIELDNAME=EMERGCONTACT, ALIAS=ENAME,     FORMAT=A35,                $
 FIELDNAME=RELATIONSHIP, ALIAS=REL,       FORMAT=A8,                 $
 FIELDNAME=BIRTHDATE,     ALIAS=BDAT,     FORMAT=YMD,                $
```

PERSINFO Structure Diagram

```
SECTION 01
              STRUCTURE OF FOCUS    FILE PERSINFO ON 01/22/08 AT 16.27.24

          PERSONAL
  01      S1
 **************
 *PIN          **I
 *INCAREOF     **
 *STREETNO     **
 *APT          **
 *             **
 ***************
  **************
```

SALHIST Data Source

In this section:

SALHIST contains an information about an employee's salary history. The PIN field is indexed. Both the PIN and EFFECTDATE fields are keys.

SALHIST MASTER File

```
FILENAME=SALHIST,  SUFFIX=FOC
SEGNAME=SLHISTRY,  SEGTYPE=SH2
 FIELDNAME=PIN,         ALIAS=ID,     FORMAT=A9,      INDEX=I,  $
 FIELDNAME=EFFECTDATE, ALIAS=EDAT,    FORMAT=YMD,               $
 FIELDNAME=OLDSALARY,   ALIAS=OSAL,   FORMAT=D12.2,             $
```

SALHIST Structure Diagram

```
SECTION 01
              STRUCTURE OF FOCUS    FILE SALHIST  ON 01/22/08 AT 16.28.02

          SLHISTRY
  01      SH2
 **************
 *PIN          **I
 *EFFECTDATE   **
 *OLDSALARY    **
 *             **
 *             **
 ***************
  **************
```

PAYHIST File

The PAYHIST data source 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.

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,$
```

PAYHIST Structure Diagram

```
SECTION 01

            STRUCTURE OF FIX    FILE PAYHIST ON 05/15/03 AT 14.51.59

          PAYSEG
  01      S1
 **************
 *SOC_SEC_NO  **
 *DATE_OF_IN  **
 *AMT_OF_INC  **
 *PCT_INC     **
 *            **
 ***************
  **************
```

COMASTER File

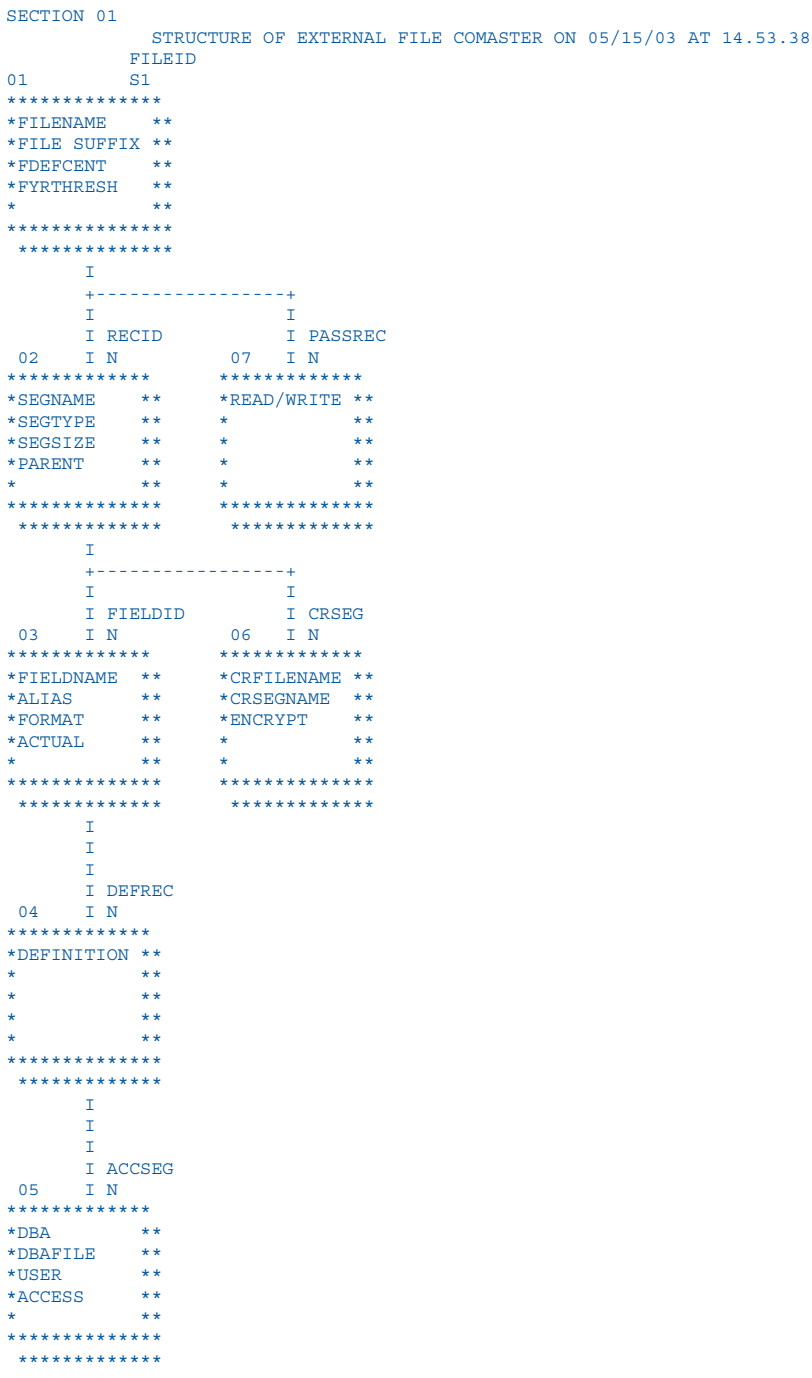
The COMASTER file is used to display the file structure and contents of each segment in a data source. Since COMASTER is used for debugging other Master Files, a corresponding FOCEXEC does not exist for the COMASTER file. Its segments are:

- ❏ FILEID, which lists file information.
- ❏ RECID, which lists segment information.
- ❏ FIELDID, which lists field information.
- ❏ DEFREC, which lists a description record.
- ❏ PASSREC, which lists read/write access.
- ❏ CRSEG, which lists cross-reference information for segments.
- ❏ ACCSEG, which lists DBA information.

COMASTER Master File

```
SUFFIX=COM,SEGNAME=FILEID
 FIELDNAME=FILENAME     ,FILE          ,A8 ,       ,$
 FIELDNAME=FILE SUFFIX ,SUFFIX        ,A8 ,       ,$
 FIELDNAME=FDEFCENT     ,FDFC          ,A4 ,       ,$
 FIELDNAME=FYRTHRESH    ,FYRT          ,A2 ,       ,$
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 ,      ,$
 FIELDNAME=DEFCENT      ,DFC           ,A4 ,       ,$
 FIELDNAME=YRTHRESH     ,YRT           ,A4 ,       ,$
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,       ,$
```

COMASTER Structure Diagram



VIDEOTRK, MOVIES, and ITEMS Data Sources

In this section:

VIDEOTRK contains sample data about customer, rental, and purchase information for a video rental business. It can be joined to the MOVIES or ITEMS data source. VIDEOTRK and MOVIES are used in examples that illustrate the use of the Maintain facility.

VIDEOTRK Master File

```
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,  $
```

VIDEOTRK Structure Diagram

```
SECTION 01

             STRUCTURE OF FOCUS    FILE VIDEOTRK ON 05/15/03 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 Master File

```
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,   $
```

MOVIES Structure Diagram

```
SECTION 01

             STRUCTURE OF FOCUS    FILE MOVIES   ON 05/15/03 AT 12.26.05

         MOVINFO
 01      S1
**************
*MOVIECODE   **I
*TITLE       **
*CATEGORY    **
*DIRECTOR    **
*            **
***************
 **************
```

ITEMS Master File

```
FILENAME=ITEMS,    SUFFIX=FOC
 SEGNAME=ITMINFO, SEGTYPE=S1
  FIELDNAME=PRODCODE,   ALIAS=PCOD,   FORMAT=A6,   INDEX=I, $
  FIELDNAME=PRODNAME,   ALIAS=PROD,   FORMAT=A20,           $
  FIELDNAME=OURCOST,    ALIAS=WCOST,  FORMAT=F6.2,          $
  FIELDNAME=RETAILPR,   ALIAS=PRICE,  FORMAT=F6.2,          $
  FIELDNAME=ON_HAND,    ALIAS=NUM,    FORMAT=I5,            $
```

ITEMS Structure Diagram

```
SECTION 01

             STRUCTURE OF FOCUS    FILE ITEMS    ON 05/15/03 AT 12.26.05

         ITMINFO
 01      S1
**************
*PRODCODE    **I
*PRODNAME    **
*OURCOST     **
*RETAILPR    **
*            **
***************
 **************
```

VIDEOTR2 Data Source

In this section:

VIDEOTR2 contains sample data about customer, rental, and purchase information for a video rental business. It consists of four segments.

VIDEOTR2 Master File

```
FILENAME=VIDEOTR2, 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=HYYMDI,      $
 SEGNAME=SALES,    SEGTYPE=S2,   PARENT=TRANSDAT
  FIELDNAME=TRANSCODE,    ALIAS=TCOD,         FORMAT=I3,          $
  FIELDNAME=QUANTITY,     ALIAS=NO,           FORMAT=I3S,         $
  FIELDNAME=TRANSTOT,     ALIAS=TTOT,         FORMAT=F7.2S,       $
 SEGNAME=RENTALS,  SEGTYPE=S2,   PARENT=TRANSDAT
  FIELDNAME=MOVIECODE,    ALIAS=MCOD,         FORMAT=A6, INDEX=I, $
  FIELDNAME=COPY,         ALIAS=COPY,         FORMAT=I2,          $
  FIELDNAME=RETURNDATE,   ALIAS=INDATE,       FORMAT=YMD,         $
  FIELDNAME=FEE,          ALIAS=FEE,          FORMAT=F5.2S,       $
```

VIDEOTR2 Structure Diagram

Gotham Grinds Data Sources

In this section:

Gotham Grinds is a group of data sources that contain sample data about a specialty items company.

- GGDEMOG contains demographic information about the customers of Gotham Grinds, a company that sells specialty items like coffee, gourmet snacks, and gifts. It consists of one segment, DEMOG01.
- GGORDER contains order information for Gotham Grinds. It consists of two segments, ORDER01 and ORDER02.
- GGPRODS contains product information for Gotham Grinds. It consists of one segment, PRODS01.
- GGSALES contains sales information for Gotham Grinds. It consists of one segment, SALES01.
- GGSTORES contains information for each of Gotham Grinds' 12 stores in the United States. It consists of one segment, STORES01.

GGDEMOG Master File

```
FILENAME=GGDEMOG, SUFFIX=FOC
 SEGNAME=DEMOG01, SEGTYPE=S1
  FIELD=ST,       ALIAS=E02, FORMAT=A02, INDEX=I,TITLE='State',
   DESC='State',$
  FIELD=HH,       ALIAS=E03, FORMAT=I09, TITLE='Number of Households',
   DESC='Number of Households',$
  FIELD=AVGHHSZ98,ALIAS=E04, FORMAT=I09, TITLE='Average Household Size',
   DESC='Average Household Size',$
  FIELD=MEDHHI98, ALIAS=E05, FORMAT=I09, TITLE='Median Household Income',
   DESC='Median Household Income',$
  FIELD=AVGHHI98, ALIAS=E06, FORMAT=I09, TITLE='Average Household
Income',
   DESC='Average Household Income',$
  FIELD=MALEPOP98,ALIAS=E07, FORMAT=I09, TITLE='Male Population',
   DESC='Male Population',$
  FIELD=FEMPOP98, ALIAS=E08, FORMAT=I09, TITLE='Female Population',
   DESC='Female Population',$
  FIELD=P15TO1998,ALIAS=E09, FORMAT=I09, TITLE='15 to 19',
   DESC='Population 15 to 19 years old',$
  FIELD=P20TO2998,ALIAS=E10, FORMAT=I09, TITLE='20 to 29',
   DESC='Population 20 to 29 years old',$
  FIELD=P30TO4998,ALIAS=E11, FORMAT=I09, TITLE='30 to 49',
   DESC='Population 30 to 49 years old',$
  FIELD=P50TO6498,ALIAS=E12, FORMAT=I09, TITLE='50 to 64',
   DESC='Population 50 to 64 years old',$
  FIELD=P65OVR98, ALIAS=E13, FORMAT=I09, TITLE='65 and over',
   DESC='Population 65 and over',$
```

GGDEMOG Structure Diagram

```
SECTION 01

             STRUCTURE OF FOCUS    FILE GGDEMOG  ON 05/15/03 AT 12.26.05

         GGDEMOG
 01      S1
**************
*ST           **I
*HH           **
*AVGHHSZ98    **
*MEDHHI98     **
*             **
***************
 **************
```

GGORDER Master File

```
FILENAME=GGORDER, SUFFIX=FOC,$
 SEGNAME=ORDER01, SEGTYPE=S1,$
  FIELD=ORDER_NUMBER, ALIAS=ORDNO1,   FORMAT=I6,  TITLE='Order,Number',
   DESC='Order Identification Number',$
  FIELD=ORDER_DATE,   ALIAS=DATE,     FORMAT=MDY, TITLE='Order,Date',
   DESC='Date order was placed',$
  FIELD=STORE_CODE,   ALIAS=STCD,     FORMAT=A5,  TITLE='Store,Code',
   DESC='Store Identification Code (for order)',$
  FIELD=PRODUCT_CODE, ALIAS=PCD,      FORMAT=A4,  TITLE='Product,Code',
   DESC='Product Identification Code (for order)',$
  FIELD=QUANTITY,     ALIAS=ORDUNITS, FORMAT=I8,  TITLE='Ordered,Units',
   DESC='Quantity Ordered',$
SEGNAME=ORDER02, SEGTYPE=KU, PARENT=ORDER01, CRFILE=GGPRODS, CRKEY=PCD,
CRSEG=PRODS01  ,$
```

GGORDER Structure Diagram

```
SECTION 01

              STRUCTURE OF FOCUS    FILE GGORDER  ON 05/15/03 AT 16.45.48

          GGORDER
  01      S1
 **************
 *ORDER_NUMBER**
 *ORDER_DATE  **
 *STORE_CODE  **
 *PRODUCT_CODE**
 *            **
 ***************
  **************
        I
        I
        I
        I ORDER02
  02    I KU
 ..............
 :PRODUCT_ID  :K
 :PRODUCT_DESC:
 :VENDOR_CODE :
 :VENDOR_NAME :
 :            :
 :............:
```

GGPRODS Master File

```
FILENAME=GGPRODS, SUFFIX=FOC
 SEGNAME=PRODS01, SEGTYPE=S1
  FIELD=PRODUCT_ID, ALIAS=PCD, FORMAT=A4, INDEX=I, TITLE='Product,Code',
   DESC='Product Identification Code',$
  FIELD=PRODUCT_DESCRIPTION, ALIAS=PRODUCT, FORMAT=A16, TITLE='Product',
   DESC='Product Name',$
  FIELD=VENDOR_CODE, ALIAS=VCD, FORMAT=A4, INDEX=I, TITLE='Vendor ID',
   DESC='Vendor Identification Code',$
  FIELD=VENDOR_NAME, ALIAS=VENDOR, FORMAT=A23, TITLE='Vendor Name',
   DESC='Vendor Name',$
  FIELD=PACKAGE_TYPE, ALIAS=PACK, FORMAT=A7, TITLE='Package',
   DESC='Packaging Style',$
  FIELD=SIZE, ALIAS=SZ, FORMAT=I2, TITLE='Size',
   DESC='Package Size',$
  FIELD=UNIT_PRICE, ALIAS=UNITPR, FORMAT=D7.2, TITLE='Unit,Price',
   DESC='Price for one unit',$
```

GGPRODS Structure Diagram

```
SECTION 01

             STRUCTURE OF FOCUS    FILE GGPRODS  ON 05/15/03 AT 12.26.05

         GGPRODS
 01      S1
**************
*PRODUCT_ID  **I
*PRODUCT_DESC**I
*VENDOR_CODE **
*VENDOR_NAME **
*            **
***************
 **************
```

GGSALES Master File

```
FILENAME=GGSALES, SUFFIX=FOC
 SEGNAME=SALES01, SEGTYPE=S1
  FIELD=SEQ_NO, ALIAS=SEQ, FORMAT=I5, TITLE='Sequence#',
   DESC='Sequence number in database',$
  FIELD=CATEGORY, ALIAS=E02, FORMAT=A11, INDEX=I, TITLE='Category',
   DESC='Product category',$
  FIELD=PCD, ALIAS=E03, FORMAT=A04, INDEX=I, TITLE='Product ID',
   DESC='Product Identification code (for sale)',$
  FIELD=PRODUCT, ALIAS=E04, FORMAT=A16, TITLE='Product',
   DESC='Product name',$
  FIELD=REGION, ALIAS=E05, FORMAT=A11, INDEX=I, TITLE='Region',
   DESC='Region code',$
  FIELD=ST, ALIAS=E06, FORMAT=A02, INDEX=I, TITLE='State',
   DESC='State',$
  FIELD=CITY, ALIAS=E07, FORMAT=A20, TITLE='City',
   DESC='City',$
  FIELD=STCD, ALIAS=E08, FORMAT=A05, INDEX=I, TITLE='Store ID',
   DESC='Store identification code (for sale)',$
  FIELD=DATE, ALIAS=E09, FORMAT=I8YYMD, TITLE='Date',
   DESC='Date of sales report',$
  FIELD=UNITS, ALIAS=E10, FORMAT=I08, TITLE='Unit Sales',
   DESC='Number of units sold',$
  FIELD=DOLLARS, ALIAS=E11, FORMAT=I08, TITLE='Dollar Sales',
   DESC='Total dollar amount of reported sales',$
  FIELD=BUDUNITS, ALIAS=E12, FORMAT=I08, TITLE='Budget Units',
   DESC='Number of units budgeted',$
  FIELD=BUDDOLLARS, ALIAS=E13, FORMAT=I08, TITLE='Budget Dollars',
   DESC='Total sales quota in dollars',$
```

GGSALES Structure Diagram

```
SECTION 01

             STRUCTURE OF FOCUS    FILE GGSALES  ON 05/15/03 AT 12.26.05

         GGSALES
 01      S1
**************
*SEQ_NO       **
*CATEGORY     **I
*PCD          **I
*PRODUCT      **I
*             **
***************
 **************
```

GGSTORES Master File

```
FILENAME=GGSTORES, SUFFIX=FOC
 SEGNAME=STORES01, SEGTYPE=S1
  FIELD=STORE_CODE, ALIAS=E02, FORMAT=A05, INDEX=I, TITLE='Store ID',
   DESC='Franchisee ID Code',$
  FIELD=STORE_NAME, ALIAS=E03, FORMAT=A23, TITLE='Store Name',
   DESC='Store Name',$
  FIELD=ADDRESS1, ALIAS=E04, FORMAT=A19, TITLE='Contact',
   DESC='Franchisee Owner',$
  FIELD=ADDRESS2, ALIAS=E05, FORMAT=A31, TITLE='Address',
   DESC='Street Address',$
  FIELD=CITY, ALIAS=E06, FORMAT=A22, TITLE='City',
   DESC='City',$
  FIELD=STATE, ALIAS=E07, FORMAT=A02, INDEX=I, TITLE='State',
   DESC='State',$
  FIELD=ZIP, ALIAS=E08, FORMAT=A06, TITLE='Zip Code',
   DESC='Postal Code',$
```

GGSTORES Structure Diagram

```
SECTION 01

             STRUCTURE OF FOCUS    FILE GGSTORES ON 05/15/03 AT 12.26.05

         GGSTORES
 01      S1
**************
*STORE_CODE  **I
*STORE_NAME  **
*ADDRESS1    **
*ADDRESS2    **
*            **
***************
 **************
```

Century Corp Data Sources

In this section:

Century Corp is a consumer electronics manufacturer that distributes products through retailers around the world. Century Corp has thousands of employees in plants, warehouses, and offices worldwide. Their mission is to provide quality products and services to their customers.

Century Corp is a group of data sources that contain financial, human resources, inventory, and order information. The last three data sources are designed to be used with chart of accounts data.

- CENTCOMP contains location information for stores. It consists of one segment, COMPINFO.
- CENTFIN contains financial information. It consists of one segment, ROOT_SEG.

- ❏ CENTHR contains human resources information. It consists of one segment, EMPSEG.
- ❏ CENTINV contains inventory information. It consists of one segment, INVINFO.
- ❏ CENTORD contains order information. It consists of four segments, OINFO, STOSEG, PINFO, and INVSEG.
- ❏ CENTQA contains problem information. It consists of three segments, PROD_SEG, INVSEG, and PROB_SEG.
- ❏ CENTGL contains a chart of accounts hierarchy. The field GL_ACCOUNT_PARENT is the parent field in the hierarchy. The field GL_ACCOUNT is the hierarchy field. The field GL_ACCOUNT_CAPTION can be used as the descriptive caption for the hierarchy field.
- ❏ CENTSYSF contains detail-level financial data. CENTSYSF uses a different account line system (SYS_ACCOUNT), which can be joined to the SYS_ACCOUNT field in CENTGL. Data uses "natural" signs (expenses are positive, revenue negative).
- ❏ CENTSTMT contains detail-level financial data and a cross-reference to the CENTGL data source.

CENTCOMP Master File

```
FILE=CENTCOMP, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=COMPINFO, SEGTYPE=S1, $
  FIELD=STORE_CODE, ALIAS=SNUM, FORMAT=A6, INDEX=I,
   TITLE='Store Id#:',
   DESCRIPTION='Store Id#', $
  FIELD=STORENAME, ALIAS=SNAME, FORMAT=A20,
   WITHIN=STATE,
   TITLE='Store,Name:',
   DESCRIPTION='Store Name', $
  FIELD=STATE, ALIAS=STATE, FORMAT=A2,
   WITHIN=PLANT,
   TITLE='State:',
   DESCRIPTION=State, $
  DEFINE REGION/A5=DECODE STATE ('AL' 'SOUTH' 'AK' 'WEST' 'AR' 'SOUTH'
  'AZ' 'WEST' 'CA' 'WEST' 'CO' 'WEST' 'CT' 'EAST'
  'DE' 'EAST' 'DC' 'EAST' 'FL' 'SOUTH' 'GA' 'SOUTH' 'HI' 'WEST'
  'ID' 'WEST' 'IL' 'NORTH' 'IN' 'NORTH' 'IA' 'NORTH'
  'KS' 'NORTH' 'KY' 'SOUTH' 'LA' 'SOUTH' 'ME' 'EAST' 'MD' 'EAST'
  'MA' 'EAST' 'MI' 'NORTH' 'MN' 'NORTH' 'MS' 'SOUTH' 'MT' 'WEST'
  'MO' 'SOUTH' 'NE' 'WEST'  'NV' 'WEST' 'NH' 'EAST' 'NJ' 'EAST'
  'NM' 'WEST' 'NY' 'EAST' 'NC' 'SOUTH' 'ND' 'NORTH' 'OH' 'NORTH'
  'OK' 'SOUTH' 'OR' 'WEST' 'PA' 'EAST' 'RI' 'EAST' 'SC' 'SOUTH'
  'SD' 'NORTH' 'TN' 'SOUTH' 'TX' 'SOUTH' 'UT' 'WEST' 'VT' 'EAST'
  'VA' 'SOUTH' 'WA' 'WEST' 'WV' 'SOUTH' 'WI' 'NORTH' 'WY' 'WEST'
  'NA' 'NORTH' 'ON' 'NORTH' ELSE ' ');,
   TITLE='Region:',
   DESCRIPTION=Region, $
```

CENTCOMP Structure Diagram

```
SECTION 01

            STRUCTURE OF FOCUS    FILE CENTCOMP ON 05/15/03 AT 10.20.49

          COMPINFO
  01      S1
 **************
 *STORE_CODE  **I
 *STORENAME   **
 *STATE       **
 *            **
 *            **
 ***************
  **************
```

CENTFIN Master File

```
FILE=CENTFIN, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=ROOT_SEG, SEGTYPE=S4, $
  FIELD=YEAR, ALIAS=YEAR, FORMAT=YY,
   WITHIN='*Time Period', $
  FIELD=QUARTER, ALIAS=QTR, FORMAT=Q,
   WITHIN=YEAR,
   TITLE=Quarter,
   DESCRIPTION=Quarter, $
  FIELD=MONTH, ALIAS=MONTH, FORMAT=M,
   TITLE=Month,
   DESCRIPTION=Month, $
  FIELD=ITEM, ALIAS=ITEM, FORMAT=A20,
   TITLE=Item,
   DESCRIPTION=Item, $
  FIELD=VALUE, ALIAS=VALUE, FORMAT=D12.2,
   TITLE=Value,
   DESCRIPTION=Value, $
  DEFINE ITYPE/A12=IF EDIT(ITEM,'9$$$$$$$$$$$$$$$$$$$') EQ 'E'
   THEN 'Expense' ELSE IF EDIT(ITEM,'9$$$$$$$$$$$$$$$$$$$') EQ 'R'
   THEN 'Revenue' ELSE 'Asset';,
   TITLE=Type,
   DESCRIPTION='Type of Financial Line Item',$
  DEFINE MOTEXT/MT=MONTH;,$
```

CENTFIN Structure Diagram

```
SECTION 01

            STRUCTURE OF FOCUS    FILE CENTFIN  ON 05/15/03 AT 10.25.52

          ROOT_SEG
  01      S4
 **************
 *YEAR         **
 *QUARTER      **
 *MONTH        **
 *ITEM         **
 *             **
 ***************
  **************
```

CENTHR Master File

```
FILE=CENTHR, SUFFIX=FOC,
 SEGNAME=EMPSEG, SEGTYPE=S1, $
  FIELD=ID_NUM, ALIAS=ID#, FORMAT=I9,
   TITLE='Employee,ID#',
   DESCRIPTION='Employee Indentification Number', $
  FIELD=LNAME, ALIAS=LN, FORMAT=A14,
   TITLE='Last,Name',
   DESCRIPTION='Employeee Last Name', $
  FIELD=FNAME, ALIAS=FN, FORMAT=A12,
   TITLE='First,Name',
   DESCRIPTION='Employee First Name', $
  FIELD=PLANT, ALIAS=PLT, FORMAT=A3,
   TITLE='Plant,Location',
   DESCRIPTION='Location of the manufacturing plant',
   WITHIN='*Location', $
  FIELD=START_DATE, ALIAS=SDATE, FORMAT=YYMD,
   TITLE='Starting,Date',
   DESCRIPTION='Date of employment',$
  FIELD=TERM_DATE, ALIAS=TERM_DATE, FORMAT=YYMD,
   TITLE='Termination,Date',
   DESCRIPTION='Termination Date', $
  FIELD=STATUS, ALIAS=STATUS, FORMAT=A10,
   TITLE='Current,Status',
   DESCRIPTION='Job Status', $
  FIELD=POSITION, ALIAS=JOB, FORMAT=A2,
   TITLE=Position,
   DESCRIPTION='Job Position',  $
  FIELD=PAYSCALE, ALIAS=PAYLEVEL, FORMAT=I2,
   TITLE='Pay,Level',
   DESCRIPTION='Pay Level',
   WITHIN='*Wages',$
  DEFINE POSITION_DESC/A17=IF POSITION EQ 'BM' THEN
   'Plant Manager' ELSE
   IF POSITION EQ 'MR' THEN 'Line Worker' ELSE
   IF POSITION EQ 'TM' THEN 'Line Manager' ELSE
   'Technician';
   TITLE='Position,Description',
   DESCRIPTION='Position Description',
   WITHIN='PLANT',$
  DEFINE BYEAR/YY=START_DATE;
   TITLE='Beginning,Year',
   DESCRIPTION='Beginning Year',
   WITHIN='*Starting Time Period',$
```

```
DEFINE BQUARTER/Q=START_DATE;
 TITLE='Beginning,Quarter',
 DESCRIPTION='Beginning Quarter',
 WITHIN='BYEAR',
DEFINE BMONTH/M=START_DATE;
 TITLE='Beginning,Month',
 DESCRIPTION='Beginning Month',
 WITHIN='BQUARTER',$
DEFINE EYEAR/YY=TERM_DATE;
 TITLE='Ending,Year',
 DESCRIPTION='Ending Year',
 WITHIN='*Termination Time Period',$
DEFINE EQUARTER/Q=TERM_DATE;
 TITLE='Ending,Quarter',
 DESCRIPTION='Ending Quarter',
 WITHIN='EYEAR',$
DEFINE EMONTH/M=TERM_DATE;
 TITLE='Ending,Month',
 DESCRIPTION='Ending Month',
 WITHIN='EQUARTER',$
DEFINE RESIGN_COUNT/I3=IF STATUS EQ 'RESIGNED' THEN 1
 ELSE 0;
 TITLE='Resigned,Count',
 DESCRIPTION='Resigned Count',$
DEFINE FIRE_COUNT/I3=IF STATUS EQ 'TERMINAT' THEN 1
 ELSE 0;
 TITLE='Terminated,Count',
 DESCRIPTION='Terminated Count',$
DEFINE DECLINE_COUNT/I3=IF STATUS EQ 'DECLINED' THEN 1
 ELSE 0;
 TITLE='Declined,Count',
 DESCRIPTION='Declined Count',$
DEFINE EMP_COUNT/I3=IF STATUS EQ 'EMPLOYED' THEN 1
 ELSE 0;
 TITLE='Employed,Count',
 DESCRIPTION='Employed Count',$
DEFINE PEND_COUNT/I3=IF STATUS EQ 'PENDING' THEN 1
 ELSE 0;
 TITLE='Pending,Count',
 DESCRIPTION='Pending Count',$
DEFINE REJECT_COUNT/I3=IF STATUS EQ 'REJECTED' THEN 1
 ELSE 0;
 TITLE='Rejected,Count',
 DESCRIPTION='Rejected Count',$
DEFINE FULLNAME/A28=LNAME||', '|FNAME;
 TITLE='Full Name',
 DESCRIPTION='Full Name: Last, First', WITHIN='POSITION_DESC',$
```

```
 DEFINE SALARY/D12.2=IF BMONTH LT 4 THEN PAYLEVEL * 12321
  ELSE IF BMONTH GE 4 AND BMONTH LT 8 THEN PAYLEVEL * 13827
  ELSE PAYLEVEL * 14400;,
  TITLE='Salary',
  DESCRIPTION='Salary',$
 DEFINE PLANTLNG/A11=DECODE PLANT (BOS 'Boston' DAL 'Dallas'
  LA 'Los Angeles' ORL 'Orlando' SEA 'Seattle' STL 'St Louis'
  ELSE 'n/a');$
```

CENTHR Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS    FILE CENTHR   ON 05/15/03 AT 10.40.34

          EMPSEG
  01      S1
 **************
 *ID_NUM       **
 *LNAME        **
 *FNAME        **
 *PLANT        **
 *             **
 ***************
  **************
```

CENTINV Master File

```
FILE=CENTINV, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=INVINFO, SEGTYPE=S1, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
   TITLE='Product,Number:',
   DESCRIPTION='Product Number', $
  FIELD=PRODNAME, ALIAS=PNAME, FORMAT=A30,
   WITHIN=PRODCAT,
   TITLE='Product,Name:',
   DESCRIPTION='Product Name', $
  FIELD=QTY_IN_STOCK, ALIAS=QIS, FORMAT=I7,
   TITLE='Quantity,In Stock:',
   DESCRIPTION='Quantity In Stock', $
  FIELD=PRICE, ALIAS=RETAIL, FORMAT=D10.2,
   TITLE='Price:',
   DESCRIPTION=Price, $
  FIELD=COST, ALIAS=OUR_COST, FORMAT=D10.2,
   TITLE='Our,Cost:',
   DESCRIPTION='Our Cost:', $
  DEFINE PRODCAT/A22 = IF PRODNAME CONTAINS 'LCD'
   THEN 'VCRs' ELSE IF PRODNAME
   CONTAINS 'DVD' THEN 'DVD' ELSE IF PRODNAME CONTAINS 'Camcor'
   THEN 'Camcorders'
   ELSE IF PRODNAME CONTAINS 'Camera' THEN 'Cameras' ELSE IF PRODNAME
   CONTAINS 'CD' THEN 'CD Players'
   ELSE IF PRODNAME CONTAINS 'Tape' THEN 'Digital Tape Recorders'
   ELSE IF PRODNAME CONTAINS 'Combo' THEN 'Combo Players'
   ELSE 'PDA Devices'; WITHIN=PRODTYPE, TITLE='Product Cateogory:' ,$
  DEFINE PRODTYPE/A19 = IF PRODNAME CONTAINS 'Digital' OR 'DVD' OR 'QX'
   THEN 'Digital' ELSE 'Analog';,WITHIN='*Product Dimension',
   TITLE='Product Type:',$
```

CENTINV Structure Diagram

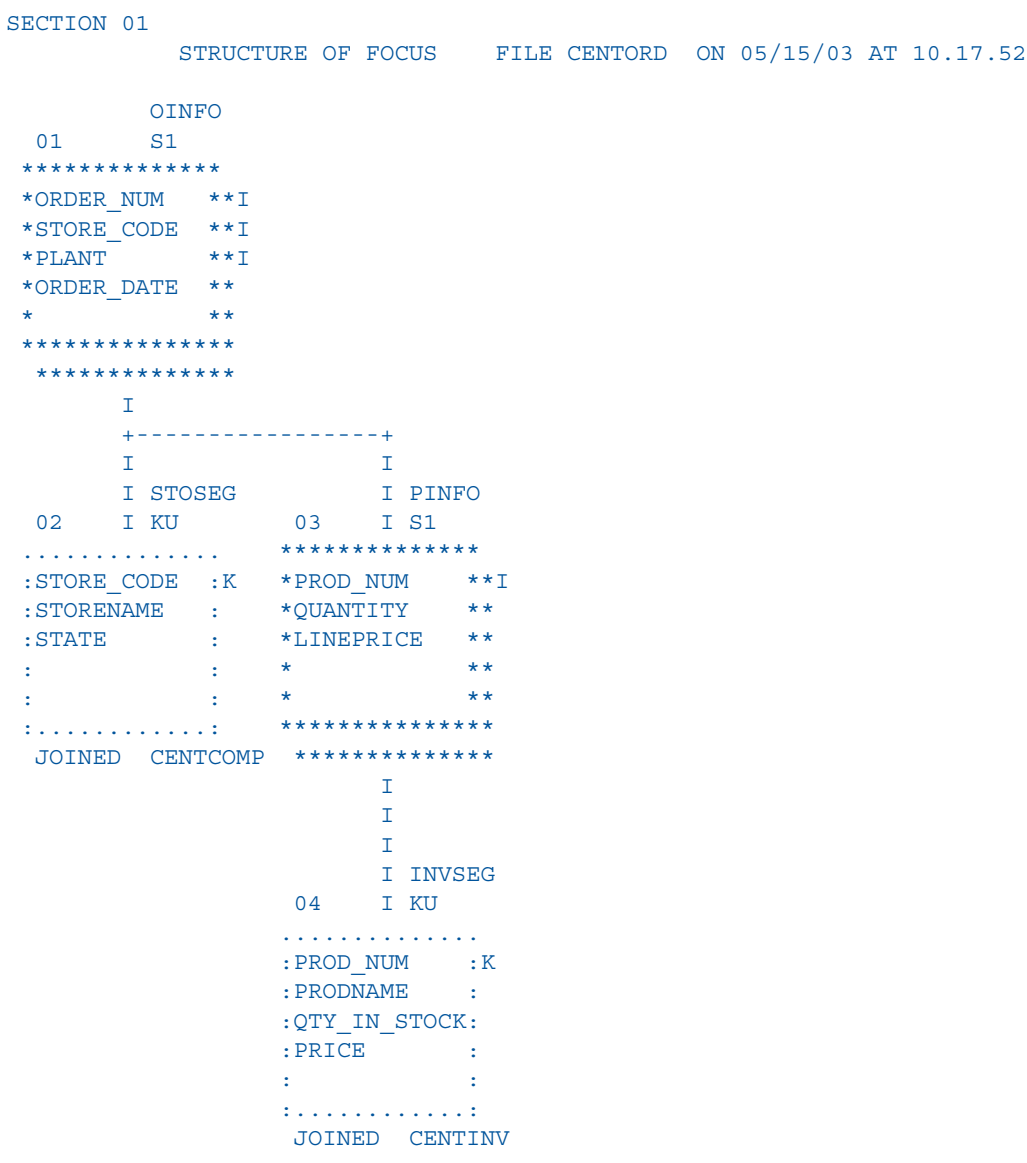
```
SECTION 01
            STRUCTURE OF FOCUS    FILE CENTINV  ON 05/15/03 AT 10.43.35

          INVINFO
  01      S1
 **************
 *PROD_NUM    **I
 *PRODNAME    **
 *QTY_IN_STOCK**
 *PRICE       **
 *            **
 ***************
  **************
```

CENTORD Master File

```
FILE=CENTORD, SUFFIX=FOC,
 SEGNAME=OINFO, SEGTYPE=S1, $
  FIELD=ORDER_NUM, ALIAS=ONUM, FORMAT=A5, INDEX=I,
   TITLE='Order,Number:',
   DESCRIPTION='Order Number', $
  FIELD=ORDER_DATE, ALIAS=ODATE, FORMAT=YYMD,
   TITLE='Date,Of Order:',
   DESCRIPTION='Date Of Order', $
  FIELD=STORE_CODE, ALIAS=SNUM, FORMAT=A6, INDEX=I,
   TITLE='Company ID#:',
   DESCRIPTION='Company ID#', $
  FIELD=PLANT, ALIAS=PLNT, FORMAT=A3, INDEX=I,
   TITLE='Manufacturing,Plant',
   DESCRIPTION='Location Of Manufacturing Plant',
   WITHIN='*Location',$
  DEFINE YEAR/YY=ORDER_DATE;,
   WITHIN='*Time Period',$
  DEFINE QUARTER/Q=ORDER_DATE;,
   WITHIN='YEAR',$
  DEFINE MONTH/M=ORDER_DATE;,
   WITHIN='QUARTER',$
 SEGNAME=PINFO, SEGTYPE=S1, PARENT=OINFO, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4,INDEX=I,
   TITLE='Product,Number#:',
   DESCRIPTION='Product Number#', $
  FIELD=QUANTITY, ALIAS=QTY, FORMAT=I8C,
   TITLE='Quantity:',
   DESCRIPTION=Quantity, $
  FIELD=LINEPRICE, ALIAS=LINETOTAL, FORMAT=D12.2MC,
   TITLE='Line,Total',
   DESCRIPTION='Line Total', $
  DEFINE LINE_COGS/D12.2=QUANTITY*COST;,
   TITLE='Line,Cost Of,Goods Sold',
   DESCRIPTION='Line cost of goods sold', $
  DEFINE PLANTLNG/A11=DECODE PLANT (BOS 'Boston' DAL 'Dallas'
   LA 'Los Angeles' ORL 'Orlando' SEA 'Seattle' STL 'St Louis'
   ELSE 'n/a');
 SEGNAME=INVSEG, SEGTYPE=DKU, PARENT=PINFO, CRFILE=CENTINV,
  CRKEY=PROD_NUM,   CRSEG=INVINFO,$
 SEGNAME=STOSEG, SEGTYPE=DKU, PARENT=OINFO, CRFILE=CENTCOMP,
  CRKEY=STORE_CODE, CRSEG=COMPINFO,$
```

CENTORD Structure Diagram



CENTQA Master File

```
FILE=CENTQA, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=PROD_SEG, SEGTYPE=S1, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
   TITLE='Product,Number',
   DESCRIPTION='Product Number', $
 SEGNAME=PROB_SEG, PARENT=PROD_SEG, SEGTYPE=S1, $
  FIELD=PROBNUM, ALIAS=PROBNO, FORMAT=I5,
   TITLE='Problem,Number',
   DESCRIPTION='Problem Number',
   WITHIN=PLANT,$
  FIELD=PLANT, ALIAS=PLT, FORMAT=A3, INDEX=I,
   TITLE=Plant,
   DESCRIPTION=Plant,
   WITHIN=PROBLEM_LOCATION,$
  FIELD=PROBLEM_DATE, ALIAS=PDATE, FORMAT=YYMD,
   TITLE='Date,Problem,Reported',
   DESCRIPTION='Date Problem Was Reported', $
  FIELD=PROBLEM_CATEGORY, ALIAS=PROBCAT, FORMAT=A20, $
   TITLE='Problem,Category',
   DESCRIPTION='Problem Category',
   WITHIN=*Problem,$
  FIELD=PROBLEM_LOCATION, ALIAS=PROBLOC, FORMAT=A10,
   TITLE='Location,Problem,Occurred',
   DESCRIPTION='Location Where Problem Occurred',
   WITHIN=PROBLEM_CATEGORY,$
  DEFINE PROB_YEAR/YY=PROBLEM_DATE;,
   TITLE='Year,Problem,Occurred',
   DESCRIPTION='Year Problem Occurred',
   WITHIN=*Time Period,$
  DEFINE PROB_QUARTER/Q=PROBLEM_DATE;
   TITLE='Quarter,Problem,Occurred',
   DESCRIPTION='Quarter Problem Occurred',
   WITHIN=PROB_YEAR,$
  DEFINE PROB_MONTH/M=PROBLEM_DATE;
   TITLE='Month,Problem,Occurred',
   DESCRIPTION='Month Problem Occurred',
   WITHIN=PROB_QUARTER,$
  DEFINE PROBLEM_OCCUR/I5 WITH PROBNUM=1;,
   TITLE='Problem,Occurrence'
   DESCRIPTION='# of times a problem occurs',$
  DEFINE PLANTLNG/A11=DECODE PLANT (BOS 'Boston' DAL 'Dallas'
   LA 'Los Angeles' ORL 'Orlando' SEA 'Seattle' STL 'St Louis'
   ELSE 'n/a');$
 SEGNAME=INVSEG, SEGTYPE=DKU, PARENT=PROD_SEG, CRFILE=CENTINV,
  CRKEY=PROD_NUM, CRSEG=INVINFO,$
```

CENTQA Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS    FILE CENTQA   ON 05/15/03 AT 10.46.43

          PROD_SEG
  01      S1
 **************
 *PROD_NUM    **I
 *            **
 *            **
 *            **
 *            **
 ***************
  **************
        I
        +-----------------+
        I                 I
        I INVSEG          I PROB_SEG
  02    I KU        03    I S1
 ..............    **************
 :PROD_NUM    :K   *PROBNUM     **
 :PRODNAME    :    *PLANT       **I
 :QTY_IN_STOCK:    *PROBLEM_DATE**
 :PRICE       :    *PROBLEM_CAT>**
 :            :    *            **
 :............:    ***************
  JOINED  CENTINV   **************
```

CENTGL Master File

```
FILE=CENTGL ,SUFFIX=FOC
 SEGNAME=ACCOUNTS, SEGTYPE=S1
  FIELDNAME=GL_ACCOUNT, ALIAS=GLACCT, FORMAT=A7,
   TITLE='Ledger,Account', FIELDTYPE=I, $
  FIELDNAME=GL_ACCOUNT_PARENT, ALIAS=GLPAR, FORMAT=A7,
   TITLE=Parent,
   PROPERTY=PARENT_OF, REFERENCE=GL_ACCOUNT, $
  FIELDNAME=GL_ACCOUNT_TYPE, ALIAS=GLTYPE, FORMAT=A1,
   TITLE=Type,$
  FIELDNAME=GL_ROLLUP_OP, ALIAS=GLROLL, FORMAT=A1,
   TITLE=Op, $
  FIELDNAME=GL_ACCOUNT_LEVEL, ALIAS=GLLEVEL, FORMAT=I3,
   TITLE=Lev, $
  FIELDNAME=GL_ACCOUNT_CAPTION, ALIAS=GLCAP, FORMAT=A30,
   TITLE=Caption,
   PROPERTY=CAPTION, REFERENCE=GL_ACCOUNT, $
  FIELDNAME=SYS_ACCOUNT, ALIAS=ALINE, FORMAT=A6,
   TITLE='System,Account,Line', MISSING=ON, $
```

CENTGL Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS    FILE CENTGL   ON 05/15/03 AT 15.18.48

          ACCOUNTS
  01      S1
 **************
 *GL_ACCOUNT  **I
 *GL_ACCOUNT_>**
 *GL_ACCOUNT_>**
 *GL_ROLLUP_OP**
 *             **
 ***************
  **************
```

CENTSYSF Master File

```
FILE=CENTSYSF ,SUFFIX=FOC
 SEGNAME=RAWDATA ,SEGTYPE=S2
  FIELDNAME = SYS_ACCOUNT ,   ,A6     , FIELDTYPE=I,
   TITLE='System,Account,Line', $
  FIELDNAME = PERIOD      ,   ,YYM    , FIELDTYPE=I,$
  FIELDNAME = NAT_AMOUNT  ,   ,D10.0 , TITLE='Month,Actual', $
  FIELDNAME = NAT_BUDGET  ,   ,D10.0 , TITLE='Month,Budget', $
  FIELDNAME = NAT_YTDAMT  ,   ,D12.0 , TITLE='YTD,Actual',   $
  FIELDNAME = NAT_YTDBUD  ,   ,D12.0 , TITLE='YTD,Budget',   $
```

CENTSYSF Structure Diagram

```
SECTION 01

            STRUCTURE OF FOCUS    FILE CENTSYSF   ON 05/15/03 AT 15.19.27

          RAWDATA
  01      S2
 **************
 *SYS_ACCOUNT **I
 *PERIOD      **I
 *NAT_AMOUNT  **
 *NAT_BUDGET  **
 *            **
 ***************
  **************
```

CENTSTMT Master File

```
FILE=CENTSTMT, SUFFIX=FOC
SEGNAME=ACCOUNTS    ,SEGTYPE=S1
  FIELD=GL_ACCOUNT,           ALIAS=GLACCT,  FORMAT=A7,
        TITLE='Ledger,Account', FIELDTYPE=I, $
  FIELD=GL_ACCOUNT_PARENT,    ALIAS=GLPAR,   FORMAT=A7,
        TITLE=Parent,
        PROPERTY=PARENT_OF, REFERENCE=GL_ACCOUNT, $
  FIELD=GL_ACCOUNT_TYPE,      ALIAS=GLTYPE,  FORMAT=A1,
        TITLE=Type,$
  FIELD=GL_ROLLUP_OP,         ALIAS=GLROLL,  FORMAT=A1,
        TITLE=Op, $
  FIELD=GL_ACCOUNT_LEVEL,     ALIAS=GLLEVEL, FORMAT=I3,
        TITLE=Lev, $
  FIELD=GL_ACCOUNT_CAPTION,   ALIAS=GLCAP,   FORMAT=A30,
        TITLE=Caption,
        PROPERTY=CAPTION, REFERENCE=GL_ACCOUNT, $
SEGNAME=CONSOL, SEGTYPE=S1, PARENT=ACCOUNTS, $
  FIELD=PERIOD, ALIAS=MONTH, FORMAT=YYM, $
  FIELD=ACTUAL_AMT, ALIAS=AA, FORMAT=D10.0, MISSING=ON,
        TITLE='Actual', $
  FIELD=BUDGET_AMT, ALIAS=BA, FORMAT=D10.0, MISSING=ON,
        TITLE='Budget', $
  FIELD=ACTUAL_YTD, ALIAS=AYTD, FORMAT=D12.0, MISSING=ON,
        TITLE='YTD,Actual', $
  FIELD=BUDGET_YTD, ALIAS=BYTD, FORMAT=D12.0, MISSING=ON,
        TITLE='YTD,Budget', $
```

CENTSTMT Structure Diagram

```
SECTION 01

          STRUCTURE OF FOCUS     FILE  CENTSTMT   ON 11/06/03 AT 16.11.59

         ACCOUNTS

 01      S1
**************
*GL_ACCOUNT  **I
*GL_ACCOUNT_>**
*GL_ACCOUNT_>**
*GL_ROLLUP_OP**
*             **
***************
 ***************
       I
       I
       I
       I CONSOL
 02    I S1
**************
*PERIOD      **
*ACTUAL_AMT  **
*BUDGET_AMT  **
*ACTUAL_YTD  **
*             **
***************
 ***************
```

B Error Messages

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.

Topics:

Accessing Error Files

For CMS, the ERRORS files are:

- FOT004 ERRORS
- FOG004 ERRORS
- FOM004 ERRORS
- FOS004 ERRORS
- FOA004 ERRORS
- FSQLXLT ERRORS
- FOCSTY ERRORS
- FOB004 ERRORS

For z/OS, these files are the following members in the ERRORS PDS:

- FOT004
- FOG004
- FOM004
- FOS004
- FOA004
- FSQLXLT
- FOCSTY
- FOB004

Displaying Messages

To display the text and explanation for any message, issue the following query command at the FOCUS command level

```
? n
```

where:

n

Is the message number.

The message number and text appear, 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 a Non-FOCUS Data Source

This appendix describes three ways to read non-FOCUS data sources with user-written procedures.

Topics:

- Dynamic and Re-Entrant Private User Exit of the FOCSAM Interface
- User-coded Data Access Modules
- Re-Entrant VSAM Compression Exit: ZCOMP1

Dynamic and Re-Entrant Private User Exit of the FOCSAM Interface

In this section:

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 exit makes it possible to combine user-written code, devoid of any dependence on internal 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, etc. The major features of the GETPRV exit are as follows:

- Through the CONTEXT parameter, the exit supports reentrancy, providing the benefit of reduced storage requirements as well as enhanced invocation performance.
- 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.
- The user exit is dynamically called at execution time, avoiding the need to modify FOCUS after each upgrade or application of maintenance. There is no need for link-edits to FOCUS.
- An initialization call allows the exit code to perform initial housekeeping.
- The QUALIF parameter supports the options (O) OPEN file, (R) OPEN request (position), (C) CLOSE, and (F) FIN of FOCUS. These control options are in addition to the (S), (G), and (E) read options.
- The 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 data sources 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 conform to following:

1. Must be able to obtain a record, given a full value of the key. The key is presumed to be unique (direct read).
2. Must be able to obtain a record greater than or equal to a given value of the full or partial key (generic read).
3. Must be able to obtain the next logical record, starting from the current position in the file (sequential read). Successive sequential reads must return records 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 re-entrant. It must be linked AMODE 31.

Implementation

The Dynamic GETPRV exit is linked as a separate module and loaded or called from FOCUS.

Master File

The Master File for data to be accessed using this user exit is the same as the description of any other data source read by the FOCSAM Interface except that the SUFFIX must specify PRIVATE and a GROUP with ALIAS=KEY must be specified. All other READ ONLY features of the FOCSAM Interface are fully supported. For example,

```
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.

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.

For example,

```
MODNAME=pgmname,$
```

To allocate this Access File, enter:

```
//ACCESS DD DSN=access.file.pds.name,DISP=SHR
```

Calling Sequence

The user-coded retrieval routine is written as a standalone program. There are no limitations to the program name other than standard IBM rules. It is recommended 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).

DDNAME

Eight-byte character argument, posted by FOCSAM, corresponding to the FOCUS file name 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.

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 exact 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 by having the high-order bit set on in the last parameter. Make sure that your program does not scan for this high-order bit.

Work Area Control Block

Example:

Sample Assembler DSECT

Sample C Declaration Required For Invoking FFUN Function

The parameter list for the work area control block is:

EYECATCH

An eight- 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 user ID 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 data source. 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

(Four 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

(Four 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

(Four 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

(Four 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.

CSEG REGI NFIND MATCH

Reserved for Information Builders' use.

IGNOR

(Four 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

In this section:

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.

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 z/OS) 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 z/OS 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 ZCOMP1, 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 to which they point.

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

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.

Topics:

- Data Storage and Display
- Rounding in Calculations and Conversions

Data Storage and Display

In this section:

Integer Fields: Format I

Floating-Point Fields: Formats F and D

Packed Decimal Format: Format P

Example:

Storing and Displaying Values

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 being stored. Floating-point fields (formats F and D) are stored as entered and rounded for display.

In general, 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 Single-Precision	F3 F5.1	123.78 123.78	123.7800 123.7800	124 123.8
D	Floating-Point Double-Precision	D3 D5.1	123.78 123.78	123.780000000000 123.780000000000	124 123.8
P	Packed	P3 P5.1	123.78 123.78	0000124 00001238	124 123.8

Note: 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.

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.

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: Storing and Displaying Values

For floating-point fields (format F or D), 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 mainframe.

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

In this section:

DEFINE and COMPUTE

Example:

Redefining Field Formats

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 eight 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. DEFINE fields are treated like data source fields, while COMPUTE 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 data source 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).

E File Description Attribute Summary

This appendix lists the attributes supported for Master Files and Access Files for FOCUSand XFOCUS data sources.

Topics:

- ❏ Overview of File Descriptions
- ❏ Master File Attributes
- ❏ Access File Attributes
- ❏ Related Commands

Overview of File Descriptions

The description of a FOCUS file is a series of comma-separated attributes which are stored in a file which is identified as a Master File.

A Master File can refer to an Access File. The Access File provides comprehensive metadata management for FOCUS data sources. It shields end users from the complex file storage and configuration details used for access to partitioned and distributed data sources.

The Access File is a series of blank-separated attributes that describe how to locate, concatenate, join, and select the appropriate physical data files for retrieval requests against one or more FOCUS data sources.

Access Files are optional, except for retrieval requests using intelligent partitioning. One Access File can contain information for one or more Master Files.

A Business View (BV) of a Master File groups related items together to reflect an application's business logic rather than the physical position of items in the data source.

Master File Attributes

Reference:

Master File FILE Attributes

Master File Segment Attributes

Master File Field Attributes

Master File Attributes That Join Data Sources

Master File Data Security Attributes

Master File Hierarchy Attributes

The following sections list the attributes supported in a Master File.

Reference: Master File FILE Attributes

ACCESS
: Identifies the Access File for a FOCUS data source.

DATASET
: Identifies the fully-qualified physical file name.

FILENAME
: Identifies the Master File.

REMARKS|DESC[RIPTION]
: Provides descriptive text in the default language.

DESC_*ln*
: Provides descriptive text in a language identified by the language code *ln*.

SUFFIX
: Identifies the type of data source.

Reference: Master File Segment Attributes

DATASET
: Identifies the fully-qualified physical file name of a LOCATION or cross-referenced segment in a FOCUS data source.

ENCRYPT
: Causes the data values for a segment to be encoded as they are added to the database.

LOCATION
: Names a separate file where a segment of data or a text field is located.

OCCURS
: Identifies repeating fields in a non-FOCUS data source.

PARENT
: Identifies a segment's parent segment.

POSITION
: Names a field in the parent segment that acts as a place holder for repeating fields in the middle of a record.

SEGNAME
: Assigns a segment name.

SEGTYPE
: Declares the sort order and number of key fields in a segment.

Reference: Master File Field Attributes

ACCEPT

Assigns a list or range of acceptable data values to a field, or identifies a file in which acceptable values are stored. The values are used to test transaction values during data maintenance operations. For files with multiple record types described by a generalized record structure, ACCEPT assigns valid RECTYPE data values.

ACTUAL

Declares the type and length of a field in a non-FOCUS data source.

ALIAS

Assigns an alternate field name. For relational databases, ALIAS is set to the field's column name. For non-FOCUS data source containing multiple record types, ALIAS is set to the value of the record type flag.

COMPUTE

Stores a temporary field definition in the Master File for use as a COMPUTE in a request.

DEFINE

Creates a virtual field.

DESC

Provides descriptive text in the default language.

DESC_*ln*

Provides descriptive text in a language identified by the language code *ln*.

FIELDNAME

Assigns a field name.

FIELDTYPE

Identifies an indexed field.

FILTER

Defines a named boolean expression to use in WHERE or IF tests in a request.

FORMAT

Declares the type, length, and display options of a field in a FOCUS database. For other data files, declares how FOCUS is to display and use the data values.

GROUP

Describes multiple fields as a single entity.

ELEMENTS

Identifies group fields as a set of elements. Eliminates the need to specify USAGE and ACTUAL formats for the group field.

HELPMESSAGE
: Assigns a one-line text message that can be displayed at the bottom of a CRTFORM screen.

INDEX
: Same as FIELDTYPE.

LOCATION
: Names a separate file where a segment of data or a text field is located.

MAPFIELD
: Is a field in the fixed portion of a segment that contains the record type for a descendant OCCURS segment with record types.

MAPVALUE
: Associates each OCCURS segment with its associated MAPFIELD record indicator.

MISSING
: Causes FOCUS to distinguish null values from intentionally entered zeros or blanks.

ORDER
: Declares an internal counter for an OCCURS segment that associates a sequence number with each repetition.

RECTYPE
: Identifies the type of record in a non-FOCUS data source with multiple record types.

TITLE
: Provides a column heading in the default language.

TITLE_*ln*
: Provides a column heading in a language identified by the language code *ln*.

USAGE
: Same as FORMAT.

Reference: Master File Attributes That Join Data Sources

CRFILE
: Names the cross-referenced file in a Master-defined join.

CRKEY
: Names the common field in a Master-defined join.

CRSEGNAME
: Names the cross-referenced segment that contains the common field in a Master-defined join.

Reference: Master File Data Security Attributes

ACCESS
: Specifies a user's access rights.

DBA
: Establishes the administrator's password.

DBAFILE
: Names a central file that contains security information.

END
: Marks the end of the Master File.

NAME
: Identifies the restricted segment or field.

RESTRICT
: Controls Access to segments and fields

USER
: Establishes a user's password.

VALUE
: Defines the test condition.

Reference: Master File Hierarchy Attributes

PROPERTY
: Identifies a field as a parent of the hierarchy field (PROPERTY=PARENT_OF) or a caption for the hierarchy field (PROPERTY=CAPTION).

REFERENCE
: Identifies the hierarchy field.

Access File Attributes

Reference:

General Access File Attributes

MDI Access File Attributes

An Access File for a FOCUS data source can contain information for multiple Master Files.

Reference: General Access File Attributes

MASTER

Identifies a Master File.

DATA

Is the fully-qualified physical file name of a partition.

WHERE

Supplies a condition that identifies which records are stored in the associated partition.

LOCATION

Identifies a segment stored in a separate physical file. Has an associaated LOCATIONDATA attribute to supply the physical file name.

Reference: MDI Access File Attributes

MDI

Identifies the name of the MDI.

TARGET_OF

Identifies the segment that contains the measures to which the MDI points. If the measures are in multiple segments, identifies the top segment that contains measures.

DIM

Identifies a field that is a dimension. A minimum of two dimensions are required.

MAXVALUES

Is the number of distinct values the dimension field can have.

WITHIN

Defines a hierarchy of dimensions by naming a child dimension within a parent dimension.

MDIDATA

Identifies the physical name of the MDI file. Multiple DATANAMEs describe concatenated partitions.

Related Commands

The following commands apply to describing data.

CHECK
: Tests a Master File for common errors, diagrams the segments and relationships, and holds information from the Master File as a data file that you can query with the reporting language.

CREATE

Creates a new FOCUS or relational data source or erases data in an existing data source.

DECRYPT
: Decodes an encoded Master File or FOCEXEC.

DEFINE
: Creates temporary fields from expressions that use constants, mathematical and logical operators, and other real or temporary fields.

ENCRYPT
: Encodes a Master File or FOCEXEC.

FILEDEF
: Assigns a temporary name to an existing file.

JOIN
: Joins two data sources using one or more fields with common data or a test condition.

NODISP
: Is a CRTFORM field attribute that you can use to suppress password display.

RESTRICT
: Applies a new or changed DBA password to a data file.

SET MAXLRECL
: Enables FOCUS to manage extra-long records.

SET USER
: Sets a password for the FOCUS session.

SET PASS
: Sets a user password.

USE
: Specifies the name and/or location of FOCUS data sources.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

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. You can contact us through the following methods:

Mail: Documentation Services - Customer Support
Information Builders, Inc.
Two Penn Plaza
New York, NY 10121-2898

Fax: (212) 967-0460

E-mail: books_info@ibi.com

Web form: http://www.informationbuilders.com/bookstore/derf.html

Name:_____

Company:_____

Address:_____

Telephone:_____Date:_____

E-mail:_____

Comments:

Reader Comments

Information Builders, Inc. Two Penn Plaza, New York, NY 10121-2898 (212) 736-4433