

# FOCUS<sup>®</sup> for S/390<sup>®</sup>

Describing Data  
Version 7.2

Cactus, EDA/SQL, FIDEL, FOCCALC, FOCUS, FOCUS Fusion, FOCUS Vision, Hospital-Trac, Information Builders, the Information Builders logo, Parlay, PC/FOCUS, SmartMart, SmartMode, SNAPpack, TableTalk, WALDO, Web390, WebFOCUS and WorldMART are registered trademarks and EDA, iWay, and iWay Software are trademarks of Information Builders, Inc.

Acrobat and Adobe are registered trademarks of Adobe Systems Incorporated.

Allaire and JRun are trademarks of Allaire Corporation.

NOMAD is a registered trademark of Aonix.

UniVerse is a registered trademark of Ardent Software, Inc.

IRMA is a trademark of Attachmate Corporation.

Baan is a registered trademark of Baan Company N.V.

SUPRA and TOTAL are registered trademarks of Cincom Systems, Inc.

Impromptu is a registered trademark of Cognos.

Alpha, DEC, DECnet, NonStop, and VAX are registered trademarks and Tru64, OpenVMS, and VMS are trademarks of Compaq Computer Corporation.

CA-ACF2, CA-Datcom, CA-IDMS, CA-Top Secret, and Ingres are registered trademarks of Computer Associates International, Inc.

MODEL 204 and M204 are registered trademarks of Computer Corporation of America.

Paradox is a registered trademark of Corel Corporation.

StorHouse is a registered trademark of FileTek, Inc.

HP MPE/iX is a registered trademark of Hewlett Packard Corporation.

Informix is a registered trademark of Informix Software, Inc.

ACF/VTAM, AIX, AS/400, CICS, DB2, DRDA, Distributed Relational Database Architecture, IBM, MQSeries, MVS/ESA, OS/2, OS/390, OS/400, RACF, RS/6000, S/390, VM/ESA, VSE/ESA and VTAM are registered trademarks and DB2/2, HiperSpace, IMS, MVS, QMF, SQL/DS, WebSphere, z/OS and z/VM are trademarks of International Business Machines Corporation.

INTERSOLVE and Q+E are registered trademarks of INTERSOLVE.

Orbit is a registered trademark of Iona Technologies Inc.

Approach and DataLens are registered trademarks of Lotus Development Corporation.

ObjectView is a trademark of Matesys Corporation.

ActiveX, FrontPage, Microsoft, MS-DOS, PowerPoint, Visual Basic, Visual C++, Visual FoxPro, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

Teradata is a registered trademark of NCR International, Inc.

Netscape, Netscape FastTrack Server, and Netscape Navigator are registered trademarks of Netscape Communications Corporation.

CORBA is a trademark of Object Management Group, Inc.

Oracle is a registered trademark and Rdb is a trademark of Oracle Corporation.

PeopleSoft is a registered trademark of PeopleSoft, Inc.

INFOAccess is a trademark of Pioneer Systems, Inc.

Progress is a registered trademark of Progress Software Corporation.

Red Brick Warehouse is a trademark of Red Brick Systems.

SAP and SAP R/3 are registered trademarks and SAP Business Information Warehouse and SAP BW are trademarks of SAP AG.

Silverstream is a trademark of Silverstream Software.

ADABAS is a registered trademark of Software A.G.

CONNECT:Direct is a trademark of Sterling Commerce.

Java and all Java-based marks, NetDynamics, Solaris, SunOS, and iPlanet are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

PowerBuilder and Sybase are registered trademarks and SQL Server is a trademark of Sybase, Inc.

Unicode is a trademark of Unicode, Inc.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Copyright © 2001 by Information Builders, Inc. All rights reserved. This manual, or parts thereof, may not be reproduced in any form without the written permission of Information Builders, Inc.

Printed in the U.S.A.

# Preface

This documentation describes how to create the metadata for the data sources that your FOCUS procedures will access in FOCUS® Version 7.2. It is intended for database administrators, application developers, or other information technology professionals who will create the metadata used by FOCUS to access corporate data. This manual is part of the FOCUS for S/390 documentation set.

References to MVS apply to all supported versions of the OS/390, z/OS™, and MVS operating environments. References to 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 MVS (OS/390) environments.

The users' documentation for FOCUS Version 7.2 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 is organized as follows:

<b>Chapter/Appendix</b>		<b>Contents</b>
<b>1</b>	<i>Understanding a Data Source Description</i>	Introduces Master Files and explains how to use them.
<b>2</b>	<i>Identifying a Data Source</i>	Documents how to describe general aspects of a data source.
<b>3</b>	<i>Describing a Group of Fields</i>	Documents how to describe groups of related fields or segments of a data source.
<b>4</b>	<i>Describing an Individual Field</i>	Documents how to describe specific field level information of a data source.
<b>5</b>	<i>Describing a Sequential, VSAM, or ISAM Data Source</i>	Provides supplementary information specific to sequential, VSAM, and ISAM data sources.
<b>6</b>	<i>Describing a FOCUS Data Source</i>	Provides supplementary information specific to FOCUS data sources.
<b>7</b>	<i>Defining a Join in a Master File</i>	Describes how to create a new relationship between any two segments that have at least one field in common by joining them.
<b>8</b>	<i>Checking and Changing a Master File: CHECK</i>	Describes how to use the CHECK command to validate a data source description.
<b>9</b>	<i>Accessing a FOCUS Data Source: USE</i>	Describes how to assign a logical name to a FOCUS data source.
<b>10</b>	<i>Providing Data Source Security: DBA</i>	Describes how to control access to a data source by adding security attributes to the data source description.
<b>A</b>	<i>Master Files and Diagrams</i>	Contains Master Files and diagrams of sample data sources used in the documentation examples.
<b>B</b>	<i>Error Messages</i>	Describes how to access information about error messages.
<b>C</b>	<i>User Exits for a Non-FOCUS Data Source</i>	Describes how to read non-FOCUS data sources with user-written procedures.
<b>D</b>	<i>Rounding in FOCUS</i>	Describes how FOCUS rounds numbers for each numeric data type.

# Summary of New Features

The new FOCUS features and enhancements described in this documentation set are listed in the following table.

<b>New Feature</b>	<b>Manual</b>	<b>Chapter</b>
Field-based Reformatting	<i>Creating Reports</i>	Chapter 1, <i>Creating Tabular Reports</i>
Increased Report Width	<i>Creating Reports</i>	Chapter 1, <i>Creating Tabular Reports</i>
ACROSS-TOTAL	<i>Creating Reports</i>	Chapter 4, <i>Sorting Tabular Reports</i>
Tiles	<i>Creating Reports</i>	Chapter 4, <i>Sorting Tabular Reports</i>
DEFINE FILE SAVE and DEFINE FILE RETURN	<i>Creating Reports</i>	Chapter 6, <i>Creating Temporary Fields</i>
Forecast	<i>Creating Reports</i>	Chapter 6, <i>Creating Temporary Fields</i>
Creating Comma-Delimited Files	<i>Creating Reports</i>	Chapter 11, <i>Saving and Reusing Report Output</i>
Creating Tab-Delimited Files	<i>Creating Reports</i>	Chapter 11, <i>Saving and Reusing Report Output</i>
Long Master File Names	<i>Creating Reports</i>	Chapter 11, <i>Saving and Reusing Report Output</i>
JOIN WHERE	<i>Creating Reports</i>	Chapter 13, <i>Joining Data Sources</i>
KEEPDEFINES	<i>Creating Reports</i>	Chapter 13, <i>Joining Data Sources</i>
Long Master File Names	<i>Describing Data</i>	Chapter 1, <i>Understanding a Data Source Description</i>
4K Alpha Fields	<i>Describing Data</i>	Chapter 4, <i>Describing an Individual Field</i>
Extended Currency Symbol Support	<i>Describing Data</i>	Chapter 4, <i>Describing an Individual Field</i>

<b>New Feature</b>	<b>Manual</b>	<b>Chapter</b>
SUFFIX = COMT/COMMA/TABT	<i>Describing Data</i>	Chapter 5, <i>Describing a Sequential, VSAM, or ISAM Data Source</i>
AUTODATE	<i>Describing Data</i>	Chapter 6, <i>Describing a FOCUS Data Source</i>
CDN	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
CENT-ZERO	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
Exit on Error	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
KEEPDEFINES	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
PCOMMA	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
Unlimited -INCLUDEs	<i>Developing Applications</i>	Chapter 2, <i>Managing an Application With Dialogue Manager</i>
SQUEEZ Function	<i>Using Functions</i>	Chapter 3, <i>Character Functions</i>
STRIP Function	<i>Using Functions</i>	Chapter 3, <i>Character Functions</i>
TRIM Function	<i>Using Functions</i>	Chapter 3, <i>Character Functions</i>
DYNAM ALLOC LONGNAME	<i>Overview and Operating Environments</i>	Chapter 5, <i>OS/390 and MVS Guide to Operations</i>

# Documentation Conventions

The following conventions apply throughout this manual:

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

## Related Publications

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

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

## Customer Support

Do you have questions about FOCUS?

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

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our World Wide Web site, <http://www.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](http://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 number (xxxx.xx).
- The FOCEXEC procedure (preferably with line numbers).
- Master File with picture (provided by CHECK FILE).



- Run sheet (beginning at login, including call to FOCUS), containing the following information:
  - ? RELEASE
  - ? FDT
  - ? LET
  - ? LOAD
  - ? COMBINE
  - ? JOIN
  - ? DEFINE
  - ? STAT
  - ? SET
  - ? SET GRAPH
  - ? USE
  - For MVS, ? TSO DDNAME
  - For VM, CMS QFI
- The exact nature of the problem:
  - Are the results or the format incorrect; are the text or calculations missing or misplaced?
  - The error message and code, if applicable.
  - Is this related to any other problem?
- Has the procedure or query ever worked in its present form? Has it been changed recently? How often does the problem occur?
- What release of the operating system are you using? Has it, FOCUS, your security system, or an interface system changed?
- Is this problem reproducible? If so, how?
- Have you tried to reproduce your problem in the simplest form possible? For example, if you are having problems joining two databases, have you tried executing a query containing just the code to access the database?
- Do you have a trace file?
- How is the problem affecting your business? Is it halting development or production? Do you just have questions about functionality or documentation?

## User Feedback

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

# Contents

<b>1</b>	<b>Understanding a Data Source Description .....</b>	<b>1-1</b>
	A Note About Data Source Terminology .....	1-2
	What Is a Data Source Description? .....	1-2
	How an Application Uses a Data Source Description .....	1-3
	What Does a Master File Describe? .....	1-3
	Identifying a Data Source .....	1-3
	Identifying and Relating a Group of Fields .....	1-4
	Describing a Field .....	1-4
	Creating a Data Source Description .....	1-4
	Creating a Master File and Access File Using an Editor .....	1-4
	Naming a Master File .....	1-5
	Using Long Master File Names on OS/390 .....	1-5
	Member Names for Long Master File Names in OS/390 .....	1-5
	What Is in a Master File? .....	1-9
	Improving Readability .....	1-10
	Adding a Comment .....	1-11
	Editing and Validating a Master File .....	1-11
<b>2</b>	<b>Identifying a Data Source .....</b>	<b>2-1</b>
	Specifying a Data Source Name: FILENAME .....	2-2
	Identifying a Data Source Type: SUFFIX .....	2-2
	Specifying a Physical File Name: DATASET .....	2-5
	DATASET Behavior in FOCUS Data Sources .....	2-5
	DATASET Behavior in Fixed-Format Sequential Data Sources .....	2-8
	DATASET Behavior in VSAM Data Sources .....	2-9
<b>3</b>	<b>Describing a Group of Fields .....</b>	<b>3-1</b>
	Defining a Single Group of Fields .....	3-2
	Understanding Segments .....	3-2
	Understanding Segment Instances .....	3-3
	Understanding Segment Chains .....	3-3
	Identifying Key Fields .....	3-4
	Identifying a Segment: SEGNAME .....	3-4
	Identifying a Logical View: Redefining a Segment .....	3-5
	Relating Multiple Groups of Fields .....	3-7
	Facilities for Specifying Relationships .....	3-8
	Identifying a Parent Segment: PARENT .....	3-8
	Identifying the Type of Relationship: SEGTYPE .....	3-9

- Logical Dependence: The Parent-Child Relationship .....3-9
  - Understanding Root Segments .....3-12
  - Understanding Descendant Segments .....3-12
  - Understanding Ancestral Segments.....3-13
- Logical Independence: Multiple Paths .....3-13
  - Understanding Multiple Paths .....3-14
  - Understanding Logical Independence .....3-15
- Cardinal Relationships Between Segments .....3-15
- One-to-One Relationships .....3-16
  - Where to Use One-to-One Relationships .....3-18
  - Implementing One-to-One Relationships in Relational Data Sources .....3-18
  - Implementing One-to-One Relationships in Sequential Data Sources .....3-18
  - Implementing One-to-One Relationships in FOCUS Data Sources .....3-18
- One-to-Many Relationships.....3-19
  - Implementing One-to-Many Relationships in Relational Data Sources.....3-20
  - Implementing One-to-Many Relationships in VSAM and Sequential Data Sources .....3-21
  - Implementing One-to-Many Relationships in FOCUS Data Sources .....3-21
- Many-to-Many Relationships .....3-22
  - Implementing Many-to-Many Directly .....3-22
  - Implementing Many-to-Many Indirectly.....3-23
- Recursive Relationships .....3-27
- Relating Segments From Different Types of Data Sources.....3-30
- Rotating a Data Source: Alternate Views .....3-31
  - Other Uses of an Alternate View.....3-32
- 4 Describing An Individual Field .....4-1**
  - Field Characteristics .....4-2
  - The Field's Name: FIELDNAME .....4-3
    - Using a Long and Qualified Field Name.....4-4
    - Using a Duplicate Field Name .....4-6
    - Rules for Evaluating a Qualified Field Name .....4-7
  - The Field's Synonym: ALIAS.....4-10
    - Implementing a Field Synonym .....4-11
  - The Displayed Data Type: USAGE.....4-12
    - Data Type Formats .....4-13
    - Integer Format.....4-14
    - Floating-Point Double-Precision Format.....4-14
    - Floating-Point Single-Precision Format .....4-15
    - Packed-Decimal Format .....4-15
    - Numeric Display Options .....4-16
    - Extended Currency Symbol Display Options.....4-17
    - Alphanumeric Format.....4-20

Date Formats .....	4-21
Date Display Options .....	4-22
Controlling the Date Separator .....	4-26
Date Translation .....	4-26
Using a Date Field.....	4-27
Numeric Date Literals .....	4-28
Date Fields in Arithmetic Expressions .....	4-29
Converting a Date Field .....	4-29
How a Date Field Is Represented Internally.....	4-30
Displaying a Non-Standard Date Format .....	4-31
Date Format Support .....	4-31
Alphanumeric and Numeric Formats With Date Display Options .....	4-32
Date-Time Formats .....	4-32
Describing a Date-Time Field .....	4-33
Specifying a Date-Time Value .....	4-38
Text Field Format.....	4-40
The Stored Data Type: ACTUAL .....	4-41
The ACTUAL Attribute .....	4-41
Null or MISSING Values: MISSING.....	4-44
Using a Missing Value .....	4-45
Validating Data: ACCEPT .....	4-46
Online Help Information: HELPMESSAGE.....	4-47
Setting a HELP (PF) Key .....	4-48
Alternative Report Column Titles: TITLE .....	4-49
Documenting the Field: DESCRIPTION .....	4-50
Describing a Virtual Field: DEFINE .....	4-51
Using a Virtual Field .....	4-52
<b>5 Describing a Sequential, VSAM, or ISAM Data Source .....</b>	<b>5-1</b>
Sequential Data Source Formats.....	5-2
What Is a Fixed-Format Data Source? .....	5-2
What Is a Comma or Tab-Delimited Data Source? .....	5-4
What Is a Free-Format Data Source? .....	5-5
Rules for Maintaining a Free-Format Data Source.....	5-6
Standard Master File Attributes for a Sequential Data Source .....	5-6
Standard Master File Attributes for a VSAM or ISAM Data Source .....	5-7
Describing a Group Field .....	5-7
Describing a Multiply Occurring Field in a Free-Format Data Source .....	5-9

Describing a Multiply Occurring Field in a Fixed-Format, VSAM, or ISAM Data Source.....	5-11
Using the OCCURS Attribute .....	5-12
Describing a Parallel Set of Repeating Fields .....	5-14
Describing a Nested Set of Repeating Fields .....	5-15
Using the POSITION Attribute.....	5-17
Specifying the ORDER Field.....	5-19
Redefining a Field in a Non-FOCUS Data Source .....	5-20
Extra-Large Record Length Support .....	5-21
Describing Multiple Record Types.....	5-22
Describing a RECTYPE Field.....	5-23
Describing Positionally Related Records .....	5-24
Ordering of Records in the Data Source.....	5-25
Describing Unrelated Records.....	5-27
Using a Generalized Record Type.....	5-31
Using an ALIAS in a Report Request .....	5-34
Combining Multiply Occurring Fields and Multiple Record Types.....	5-35
Describing a Multiply Occurring Field and Multiple Record Types.....	5-35
Describing a VSAM Repeating Group With RECTYPES.....	5-37
Describing a Repeating Group Using MAPFIELD.....	5-38
Establishing VSAM Data and Index Buffers.....	5-41
Using a VSAM Alternate Index .....	5-42
Describing a Token-Delimited Data Source.....	5-45
Reading a Complex Data Source With a User-Written Procedure .....	5-48
<b>6 Describing a FOCUS Data Source .....</b>	<b>6-1</b>
Designing a FOCUS Data Source .....	6-2
Data Relationships.....	6-2
Join Considerations .....	6-3
General Efficiency Considerations.....	6-3
Changing a FOCUS Data Source .....	6-4
Describing a Single Segment.....	6-5
Maximum Number of Segments .....	6-5
Describing Keys, Sort Order, and Segment Relationships: SEGTYPE.....	6-5
Describing a Key Field.....	6-7
Describing Sort Order .....	6-8
Understanding Sort Order .....	6-8
Describing Segment Relationships.....	6-9
Storing a Segment in a Different Location: LOCATION .....	6-9
Separating Large Text Fields .....	6-11
Limits on the Number of LOCATION Files .....	6-12
Timestamping a FOCUS Segment: AUTODATE.....	6-12

Describing an Individual Field .....	6-15
The ACCEPT Attribute .....	6-15
The INDEX Attribute .....	6-16
Joins and the INDEX Attribute .....	6-17
FORMAT and MISSING: Internal Storage Requirements.....	6-18
Describing Two-Gigabyte and Partitioned FOCUS Data Sources .....	6-19
Partitioning a FOCUS Data Source .....	6-19
Intelligent Partitioning.....	6-20
Specifying an Access File in a FOCUS Master File.....	6-21
The FOCUS Access File .....	6-22
FOCUS Access File Attributes.....	6-22
Describing Joined Data Sources.....	6-26
<b>7 Defining a Join in a Master File .....</b>	<b>7-1</b>
Join Types.....	7-2
Static Joins Defined in the Master File: SEGTYPE = KU and KM .....	7-3
Describing a Unique Join: SEGTYPE = KU .....	7-3
Using a Unique Join for Decoding .....	7-6
Describing a Non-Unique Join: SEGTYPE = KM .....	7-7
Using Cross-Referenced Descendant Segments: SEGTYPE = KL and KLU .....	7-10
Hierarchies of Linked Segments .....	7-14
Dynamic Joins Defined in the Master File: SEGTYPE = DKU and DKM.....	7-15
Comparing Static and Dynamic Master File Defined Joins and the JOIN Command.....	7-17
Joining to One Cross-Referenced Segment From Several Host Segments.....	7-19
Joining From Several Segments in One Host Data Source .....	7-19
Joining From Several Segments in Several Host Data Sources: Multiple Parents .....	7-22
Recursive Reuse of a Segment .....	7-23
<b>8 Checking and Changing a Master File: CHECK .....</b>	<b>8-1</b>
CHECK Command Display .....	8-3
Determining Common Errors .....	8-4
The PICTURE Option .....	8-5
The HOLD Option.....	8-8
Specifying AS Names With the HOLD Option.....	8-10
TITLE, HELPMESSAGE, and TAG Attributes .....	8-10
Virtual Fields in the Master File.....	8-11
<b>9 Accessing a FOCUS Data Source: USE .....</b>	<b>9-1</b>
The USE Command.....	9-2
Specifying a Non-Default File ID .....	9-5
Identifying New Data Sources to FOCUS.....	9-6
Accessing Data Sources in Read Only Mode .....	9-8

- Concatenating Data Sources ..... 9-9
- Accessing Simultaneous Usage Data Sources ..... 9-12
  - Multi-Thread Configuration ..... 9-13
- Using the LOCATION Attribute ..... 9-14
- Displaying the USE Options in Effect ..... 9-14
- 10 Providing Data Source Security: DBA ..... 10-1**
  - Introduction ..... 10-2
  - Implementing Data Source Security ..... 10-3
    - Identifying the DBA: The DBA Attribute ..... 10-5
    - Including the DBA Attribute in HOLD Files ..... 10-6
    - Identifying Users With Access Rights: The USER Attribute ..... 10-9
    - Establishing User Identity ..... 10-10
  - Specifying Access Types: The ACCESS Attribute ..... 10-12
    - Types of Access ..... 10-13
  - Limiting Data Source Access: The RESTRICT Attribute ..... 10-17
    - Restricting Access to Fields and Segments ..... 10-18
    - Restricting Values ..... 10-20
      - Restricting Values a User Can Write ..... 10-22
      - Restricting Values a User Can Alter ..... 10-23
      - Restricting Both Read and Write Values ..... 10-24
  - Placing Security Information in a Central Master File ..... 10-25
    - File Naming Requirements for DBAFILE ..... 10-27
    - Connection to Existing DBA System With DBAFILE ..... 10-28
    - Combining Applications With DBAFILE ..... 10-28
    - Using Filters ..... 10-28
    - Summary of Security Attributes ..... 10-29
  - Hiding the Restriction Rules: The ENCRYPT Command ..... 10-30
    - Encrypting Data ..... 10-31
    - Performance Considerations for Encrypted Data ..... 10-31
    - Restricting Existing Files ..... 10-32
    - Displaying the Decision Table ..... 10-32
    - Setting Passwords Externally ..... 10-34
  - FOCEXEC Security ..... 10-34
    - Suppressing Password Display ..... 10-35
    - Setting Passwords in Encrypted FOCEXECs ..... 10-35
    - Defining Variable Passwords ..... 10-35
    - Encrypting and Decrypting FOCEXECs ..... 10-36
    - Locking FOCEXEC Users Out of FOCUS ..... 10-36



Program Accounting/Resource Limitation .....	10-37
Program Accounting .....	10-37
Activating a DBA User Program .....	10-38
Specifications for the User-Written Program .....	10-38
Resource Limitation .....	10-39
Usage Accounting and Security Exit Routine (UACCT) .....	10-40
Absolute File Integrity .....	10-40
<b>A Master Files and Diagrams .....</b>	<b>A-1</b>
Creating Sample Data Sources .....	A-2
The EMPLOYEE Data Source .....	A-3
The EMPLOYEE Master File .....	A-4
The EMPLOYEE Structure Diagram .....	A-5
The JOBFIL Data Source .....	A-6
The JOBFIL Master File .....	A-6
The JOBFIL Structure Diagram .....	A-6
The EDUCFIL Data Source .....	A-7
The EDUCFIL Master File .....	A-7
The EDUCFIL Structure Diagram .....	A-7
The SALES Data Source .....	A-8
The SALES Master File .....	A-8
The SALES Structure Diagram .....	A-9
The PROD Data Source .....	A-10
The PROD Master File .....	A-10
The PROD Structure Diagram .....	A-10
The CAR Data Source .....	A-11
The CAR Master File .....	A-11
The CAR Structure Diagram .....	A-12
The LEDGER Data Source .....	A-13
The LEDGER Master File .....	A-13
The LEDGER Structure Diagram .....	A-13
The FINANCE Data Source .....	A-14
The FINANCE Master File .....	A-14
The FINANCE Structure Diagram .....	A-14
The REGION Data Source .....	A-15
The REGION Master File .....	A-15
The REGION Structure Diagram .....	A-15
The COURSES Data Source .....	A-16
The COURSES Master File .....	A-16
The COURSES Structure Diagram .....	A-16

The EMPDATA Data Source .....	A-17
The EMPDATA Master File .....	A-17
The EMPDATA Structure Diagram .....	A-17
The EXPERSON Data Source .....	A-18
The EXPERSON Master File .....	A-18
The EXPERSON Structure Diagram .....	A-18
The TRAINING Data Source .....	A-19
The TRAINING Master File .....	A-19
The TRAINING Structure Diagram .....	A-19
The PAYHIST File .....	A-20
The PAYHIST Master File .....	A-20
The PAYHIST Structure Diagram .....	A-20
The COMASTER File .....	A-21
The COMASTER Master File .....	A-22
The COMASTER Structure Diagram .....	A-23
The VideoTrk and MOVIES Data Sources .....	A-24
VideoTrk Master File .....	A-24
MOVIES Master File .....	A-24
VideoTrk Structure Diagram .....	A-25
MOVIES Structure Diagram .....	A-26
The VIDEOTR2 Data Source .....	A-26
The VIDEOTR2 Master File .....	A-26
The VIDEOTR2 Access File .....	A-27
The VIDEOTR2 Structure Diagram .....	A-28
The Gotham Grinds Data Sources .....	A-29
The GGDEMOG Data Source .....	A-29
The GGORDER Data Source .....	A-30
The GGPRODS Data Source .....	A-31
The GGSALES Data Source .....	A-32
The GGSTORES Data Source .....	A-33
<b>B Error Messages .....</b>	<b>B-1</b>
Accessing Error Files .....	B-2
Displaying Messages Online .....	B-3
<b>C User Exits for a Non-FOCUS Data Source .....</b>	<b>C-1</b>
The Dynamic and Re-Entrant Private User Exit of the FOCSAM Interface .....	C-2
Functional Requirements .....	C-3
Implementation .....	C-4
The Master File .....	C-4
The Access File .....	C-4
Calling Sequence .....	C-5
Work Area Control Block .....	C-6

---

User-coded Data Access Modules .....	C-10
Re-Entrant VSAM Compression Exit: ZCOMP1 .....	C-12
Linking ZCOMP1 .....	C-12
What Happens When You Use ZCOMP1 .....	C-12
ZCOMP1 Parameter List.....	C-13
<b>D Rounding in FOCUS .....</b>	<b>D-1</b>
Data Storage and Display .....	D-2
Integer Fields: Format I.....	D-3
Floating-Point Fields: Formats F and D .....	D-3
Packed Decimal Format: Format P.....	D-4
Rounding in Calculations and Conversions.....	D-6
DEFINE and COMPUTE .....	D-9
<b>Index .....</b>	<b>I-1</b>

---

## CHAPTER 1

# Understanding a Data Source Description

### Topics:

- A Note About Data Source Terminology
- What Is a Data Source Description?
- How an Application Uses a Data Source Description
- What Does a Master File Describe?
- Creating a Data Source Description
- Naming a Master File
- What Is in a Master File?

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 and FOCUS.
- Network, such as CA-IDMS.
- Indexed, such as ISAM and VSAM.
- Sequential, both fixed-format and free-format.
- Multi-dimensional, such as Fusion.

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.

## A Note About Data Source Terminology

Different types of data sources make use of similar concepts, but refer to them 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 need 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 Step 4. 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 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?

A Master File enables you to:

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

## 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

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.

For more information about coding or specifying a Master File and Access File, see *Creating a Master File and Access File Using an Editor* on page 1-4.

## Creating a Master File and Access File Using an Editor

You can 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 need 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

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

## Using Long Master File Names on OS/390

In the OS/390 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. You need to use the following to implement Master File names longer than 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 OS/390* on page 1-7.
- 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 OS/390* on page 1-5.

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 a Long Master File Name is Implemented in OS/390* on page 1-6.

## Member Names for Long Master File Names in OS/390

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 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 OS/390.

<b>Long Name</b>	<b>Member Name</b>
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 a Long Master File Name is Implemented in OS/390**

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 OS/390**

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

where:

*ddname*

Is the one- to eight-character member name of the Master File. It must be an existing member of 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 can no longer 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 OS/390**

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 (SU) is not supported on any platform.
- The DATASET attribute is not supported in a long Master File.
- The ACCESSFILE attribute is not supported with long Master Files.
- An external index is not supported.
- The LONGNAME option of the DYNAM command may only be issued from within a FOCEXEC or RPC. It cannot be used to pre-allocate long Master Files in JCL or CLISTS on OS/390.
- Long Master Files are not designed to be edited on OS/390. 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 ? TSO DDNAME queries or USE commands (OS/390 only).
- VM is not supported.
- Fusion is not supported.

## What Is in a Master File?

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 wish. 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. Each line can be a maximum of 80 characters long.

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

You can begin each attribute assignment in any position that you wish. 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. To position text, use blank spaces, not the Tab character.

You can also include blank lines to separate declarations from each other. 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*.

---

## CHAPTER 2

# Identifying a Data Source

### Topics:

- Specifying a Data Source Name:  
FILENAME
- Identifying a Data Source Type:  
SUFFIX
- Specifying a Physical File Name:  
DATASET

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?

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.
- 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 *Working With Cross-Century Dates* in the *Developing Applications* manual.

## Specifying a Data Source Name: FILENAME

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 start with a letter, and the remaining characters can be any combination of letters, numbers, and underscores (\_).

### *Example*

#### Specifying a Data Source Name

The following example specifies the data source name EMPLOYEE:

```
FILENAME = EMPLOYEE
```

## Identifying a Data Source Type: SUFFIX

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 use the full attribute name FILESUFFIX, or the shorter form SUFFIX.

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. Techniques for identifying and using your own data access routines are described in *Appendix C, User Exits for a Non-FOCUS Data Source*.



## *Syntax*

### How to Identify a Data Source Type

[FILE]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, which represents a FOCUS data source:

```
SUFFIX = FOC
```

**Reference**      **SUFFIX Values**

The following table indicates the SUFFIX value for each data source type:

| <b>Data Source Type</b>                                | <b>SUFFIX Value</b>                                              |
|--------------------------------------------------------|------------------------------------------------------------------|
| ADABAS                                                 | ADBSIN or ADBSINX<br>ADABAS (OpenVMS EDA 2.x and FOCUS 6.x)      |
| CA-Datcom/DB                                           | DATACOM                                                          |
| CA-IDMS/DB                                             | IDMSR                                                            |
| CA-IDMS/SQL                                            | SQLIDMS                                                          |
| DB2                                                    | DB2 or SQLDS                                                     |
| Fixed-format sequential                                | FIX This value is the default.<br>PRIVATE (for FOCSAM user exit) |
| FOCUS                                                  | FOC                                                              |
| Free-format (also known as comma-delimited) sequential | COM, COMMA, COMT                                                 |
| Fusion                                                 | FUSION                                                           |
| IMS                                                    | IMS                                                              |
| Millennium                                             | CPMILL or CPMILL2 (Release 2)<br>CPMILL3 (Release 3)             |
| Model 204                                              | M204IN                                                           |
| NOMAD                                                  | NMDIN                                                            |
| Oracle                                                 | SQLORA                                                           |
| SQL/DS                                                 | SQLDS                                                            |
| SUPRA                                                  | SUPRA                                                            |
| System 2000                                            | S2K                                                              |
| Tab Delimited                                          | TABT                                                             |
| Teradata                                               | SQLDBC                                                           |
| Token Delimited                                        | DFIX                                                             |
| TOTAL                                                  | TOTIN                                                            |
| VSAM                                                   | VSAM<br>PRIVATE (for FOCSAM user exit)                           |
| Non-standard data source                               | Name of the customized data access routine.                      |

## Specifying a Physical File Name: DATASET

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

User allocation and system specific behavior is as follows:

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

### Note:

- The MODIFY FIND function does not work with the DATASET attribute. To use FIND with a data source, you must 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 FOCUS Data Sources

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

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

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

- A user's explicit allocation overrides 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 de-allocate the allocation created by DATASET.

## Syntax

### How to Use the DATASET Attribute

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

where:

*filename*

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

*sinkname*

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

In MVS, the syntax is:

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

or

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

In CMS, the syntax is:

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

## Example

### Allocating a FOCUS Data Source Using the DATASET Attribute

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

For MVS,

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

For CMS,

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

**Example****Allocating a Data Source For the FOCUS Database Server**

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

For MVS,

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

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

For CMS,

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

## DATASET Behavior in Fixed-Format Sequential Data Sources

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

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

### Syntax

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

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

where:

*filename*

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

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

- A user's explicit allocation overrides DATASET attributes.

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

### Example

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

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

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

```
.  
.  
.
```

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

```
.  
.  
.
```

## DATASET Behavior in VSAM Data Sources

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

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

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

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

### *Syntax*

#### How to Use the DATASET Attribute With VSAM Data Sources

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

where:

*filename*

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

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

- A user's explicit allocation overrides DATASET attributes.

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

**Example**

**Allocating a VSAM Data Source Using the DATASET Attribute**

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

```
FILE=EXERVSAM1, 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, $
```



---

## CHAPTER 3

# Describing a Group of Fields

### Topics:

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

In a data source, certain fields may have a one-to-one correspondence and form a group. You can relate different groups to each other. For some types of data sources you can even define a logical view of a group—that is, a subset. You identify these groups and relationships between these groups 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 3-2. Relating these groups to each other is described in *Relating Multiple Groups of Fields* on page 3-7.

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

## Defining a Single Group of Fields


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

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

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

## Understanding Segments

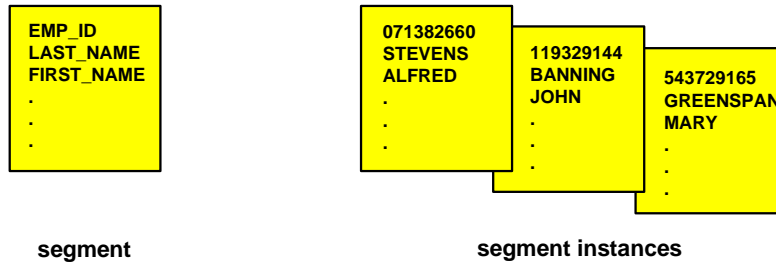
While the term *segment* may not be familiar to you, the concept behind it is universal. A segment is a group of fields that have a one-to-one correspondence with each other and usually describe a group of related characteristics. In a relational data source, a segment is equivalent to a table. Segments are the building blocks of larger data structures. You can relate different segments to each other, and describe the new structures, as described in *Relating Multiple Groups of Fields* on page 3-7.

|                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|------------------|-------------------|---|---|-------------------|-----------------|---|---|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <p><b>ID# 199329144</b><br/><b>John Banning</b></p> <p>.</p> <p>.</p> <p>.</p> <p><b>works in Production dept.</b><br/><b>earns \$29,700</b></p> <p><b>Real World</b></p> | <table border="1" style="background-color: yellow; width: 100%;"> <tr> <td><b>EMP_ID</b></td> </tr> <tr> <td><b>LAST_NAME</b></td> </tr> <tr> <td><b>FIRST_NAME</b></td> </tr> <tr> <td>.</td> </tr> <tr> <td>.</td> </tr> <tr> <td><b>DEPARTMENT</b></td> </tr> <tr> <td><b>CURR_SAL</b></td> </tr> <tr> <td>.</td> </tr> <tr> <td>.</td> </tr> <tr> <td>.</td> </tr> </table> <p><b>Segment</b></p> <p><b>(describes the real world)</b></p> | <b>EMP_ID</b> | <b>LAST_NAME</b> | <b>FIRST_NAME</b> | . | . | <b>DEPARTMENT</b> | <b>CURR_SAL</b> | . | . | . | <p><b>SEGNAME=EMPINFO</b></p> <p><b>FIELDNAME=EMP_ID, ...</b></p> <p><b>FIELDNAME=LAST_NAME, ...</b></p> <p><b>FIELDNAME=FIRST_NAME, ...</b></p> <p>.</p> <p>.</p> <p>.</p> <p><b>FIELDNAME=DEPARTMENT, ...</b></p> <p><b>FIELDNAME=CURR_SAL, ...</b></p> <p>.</p> <p>.</p> <p>.</p> |
| <b>EMP_ID</b>                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
| <b>LAST_NAME</b>                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
| <b>FIRST_NAME</b>                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
| .                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
| .                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
| <b>DEPARTMENT</b>                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
| <b>CURR_SAL</b>                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
| .                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
| .                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |
| .                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                |               |                  |                   |   |   |                   |                 |   |   |   |                                                                                                                                                                                                                                                                                      |

## Understanding Segment Instances

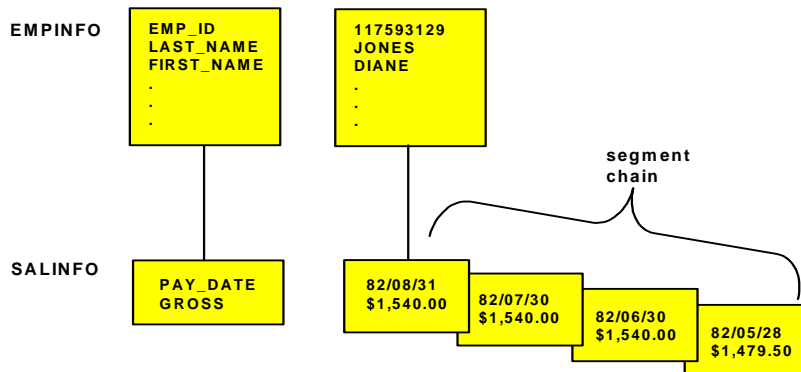
While a segment is abstract—a description of data—the segment instances that correspond to it are the actual data. Each instance is an occurrence of segment values found in the data source. For a relational data source, an instance is equivalent to a row in a table. In a single segment data source, a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

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



## Understanding Segment Chains

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



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

## Identifying Key Fields

Most segments also have key fields—that is, 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, then in the Master File, the fields constituting the primary key should be the first fields described for that segment—that is, their field declarations should come before any others in that segment.

For FOCUS data sources, you 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*. You position the key fields as the first fields in their segment.

## Identifying a Segment: SEGNAME

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

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

In a Master File, each segment name must be unique. The only exception to this rule is in a FOCUS 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 cross-referenced segments do have identical names, you can still refer to them uniquely by using the CRSEGNAME attribute. See Chapter 7, *Defining a Join in a Master File* for more information.

In a FOCUS data source, you cannot change the value of SEGNAME once 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 you want to use to identify this segment. It can be up to a maximum of eight characters long.

The first character must be a letter, and the remaining characters can be any combination of letters, numbers, and underscores (\_). Information Builders does not recommend using other characters, because they may cause problems in some operating environments or when resolving expressions.

## Example

### Identifying a Segment

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

```
SEGNAME = TICKETS
```

# Identifying a Logical View: Redefining a Segment

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 in which you include only a subset of the segment's fields (similar to a relational view), or else define the unwanted fields as one or more filler fields. This technique can be helpful if, for example, you only want to make some of the segment's fields available to an application or its users.

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

- **Relational data sources.** You can omit unwanted fields from the segment description.
- **Sequential and FOCUS data sources.** You can define unwanted fields as one or more filler fields.

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

**Example**

**Omitting Fields: Creating a Segment Subset**

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

```
FILENAME = EMPFACTS, SUFFIX = SQLORA , $
  SEGNAME = EMPFACTS, SEGTYPE = S0 , $
    FIELDNAME = EMP_NUMBER, ALIAS = ENUM, 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 could code an alternative Master File that names only the desired fields:

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

**Example**

**Redefining Fields: Creating a Filler Field**

You can 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. You would define the field's format to be alphanumeric, its length to be the number of bytes making up the underlying fields, and its name and alias to be blank. Field declarations and length are discussed in detail in Chapter 4, *Describing an Individual Field*.

For example, 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 could 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 , $
```

Note that the filler field is defined as an alphanumeric field of 29 bytes, which is the combined internal length of the fields it replaces: HIRE\_DATE (4 bytes), DEPARTMENT (10 bytes), CURR\_SAL (8 bytes), CURR\_JOBCODE (3 bytes), and ED\_HRS (4 bytes).

## Relating Multiple Groups of Fields

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

- **Describe physical relationships.** If groups of fields are already physically related in your data source, you can describe the relationship.
- **Describe logical relationships.** You can describe a logical relationship between any two segments that have at least one field in common by joining them. The underlying data structures remain physically separate, but 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.

Note that 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 need to be aware of the following:

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

## Facilities for Specifying Relationships

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.

Note that a related facility, the MATCH FILE command, enables you to implement many different types of sophisticated relationships by first describing the relationship as a series of extraction and merging conditions and then merging the related data into a new single segment data source. The result is not a joined structure but an entirely new data source that you can process 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. You 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.

Note that 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, as, for example, 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, for example, when you join several relational tables in the Master File, then you can change the PARENT attribute.

### *Syntax*

### How to Identify the Parent Segment

`PARENT = segment_name`

where:

*segment\_name*

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

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

Information Builders recommends using the PARENT attribute for unique segments with a SEGTYPE of U.



### *Example*      Identifying a Parent Segment

For example, in the EMPLOYEE data source, DEDUCT's parent is SALINFO, and 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 in different ways with different 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.

## Logical Dependence: The Parent-Child Relationship

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

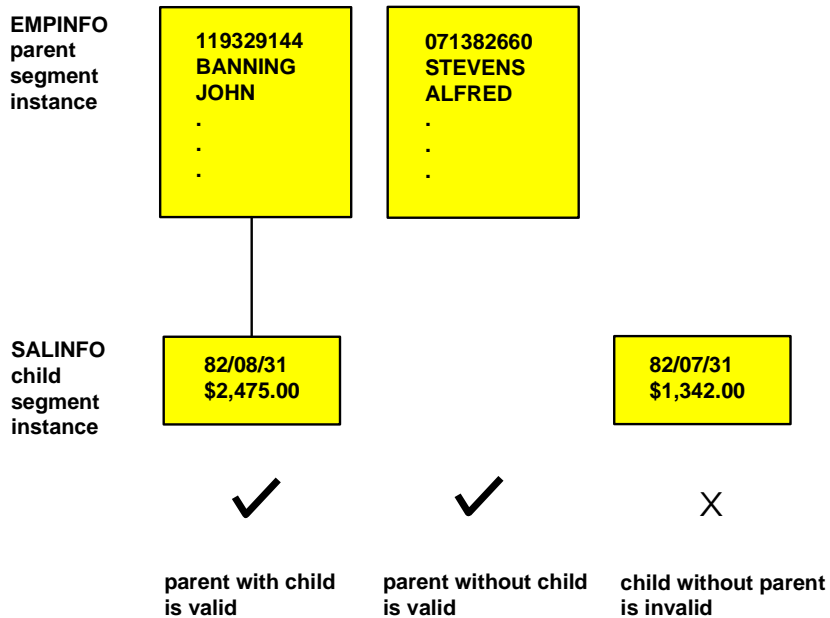
Note that if the parent-child relationship is logical and not physical—that is, if it is implemented as a join—it is possible to have a child instance without a related parent instance. In this case, the child instance will not be accessible through the join, although, of course, it will still be accessible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

If a join relates the parent and 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.

**Example**

**A Simple Parent-Child Relationship**

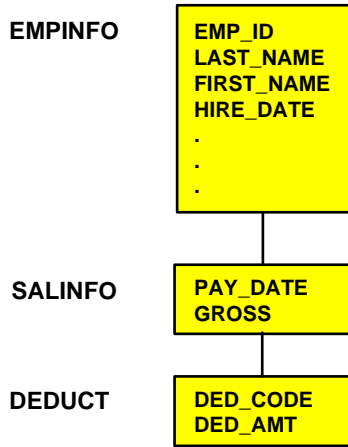
For example, in the EMPLOYEE data source, the EMPINFO and SALINFO segments are related: EMPINFO identifies an employee by ID number, while SALINFO contains the employee's pay history. EMPINFO is the parent segment, and SALINFO is a child segment dependent upon it. This relationship is illustrated by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



*Example*

### Parent-Child Relationships With Multiple Segments

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

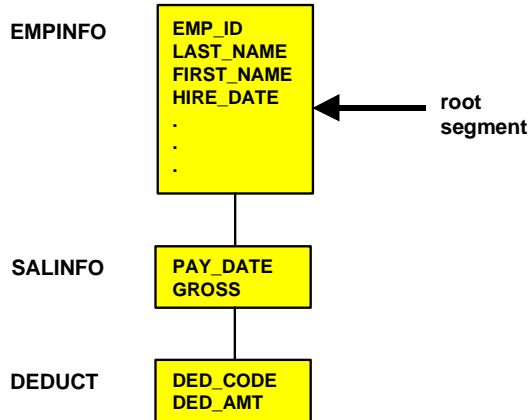


EMPINFO is related to SALINFO, and in this relationship EMPINFO is the parent segment and SALINFO is the child segment. SALINFO is also related to DEDUCT. In this second relationship, SALINFO is the parent segment and DEDUCT is the child segment. Just as SALINFO is dependent upon EMPINFO, DEDUCT is dependent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

## Understanding Root Segments

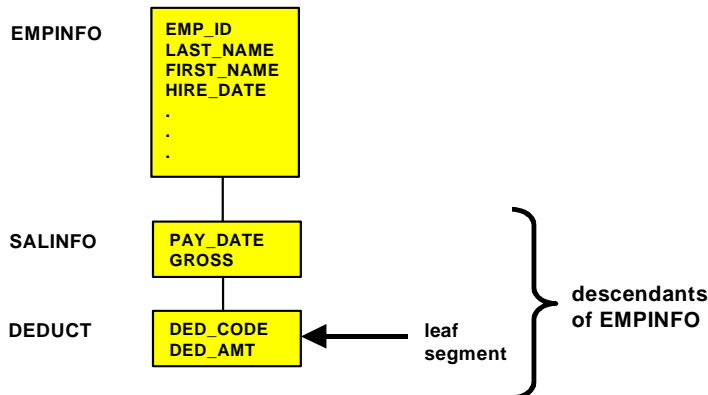
The segment that has logical precedence over the entire data structure—the parent of the entire structure—is called the *root* segment. The term *root* is used because a data structure can branch like a tree, and the root segment, like the root of a tree, is the source of the structure.

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



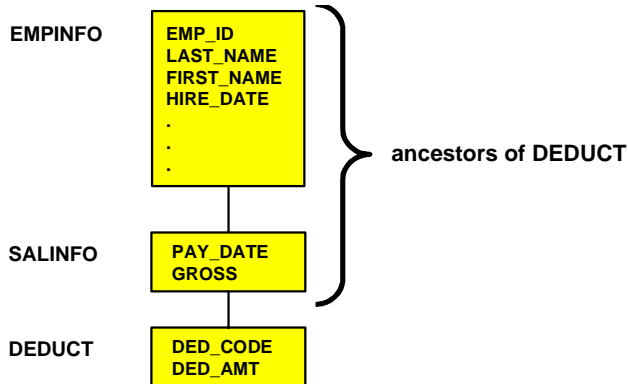
## Understanding Descendant Segments

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



## Understanding Ancestral Segments

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



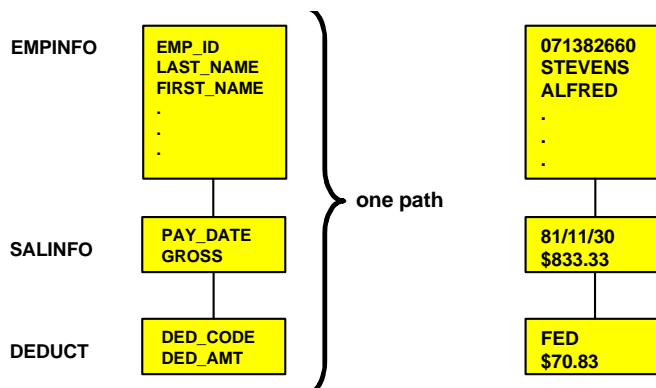
## Logical Independence: Multiple Paths

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

### Example

#### Understanding a Single Path

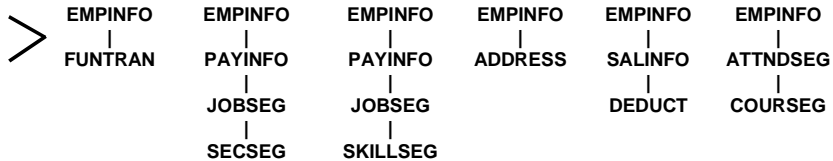
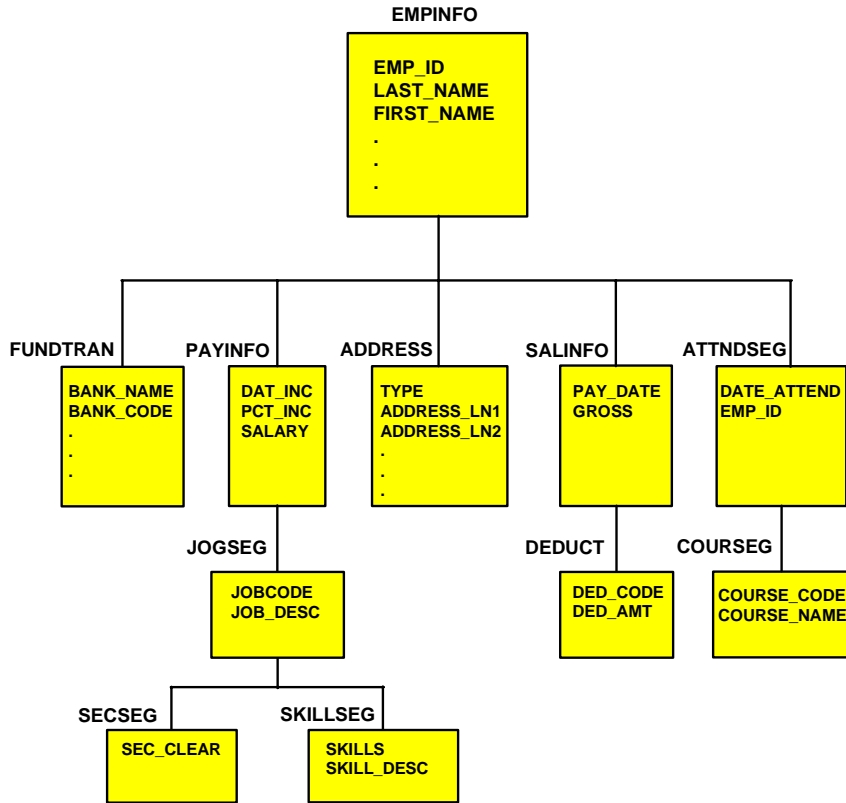
For example, in the following view of the EMPLOYEE data source, EMPINFO, SALINFO, and DEDUCT form a path. An instance of DEDUCT (paycheck deductions) can exist only if a related instance of SALINFO (the paycheck) exists, and the instance of SALINFO (the employee's paycheck) can exist only if a related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



## Understanding Multiple Paths

Now consider the full EMPLOYEE structure, which includes the EMPLOYEE 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 in a data structure is the end of a separate path.



## Understanding Logical Independence

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

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

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

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

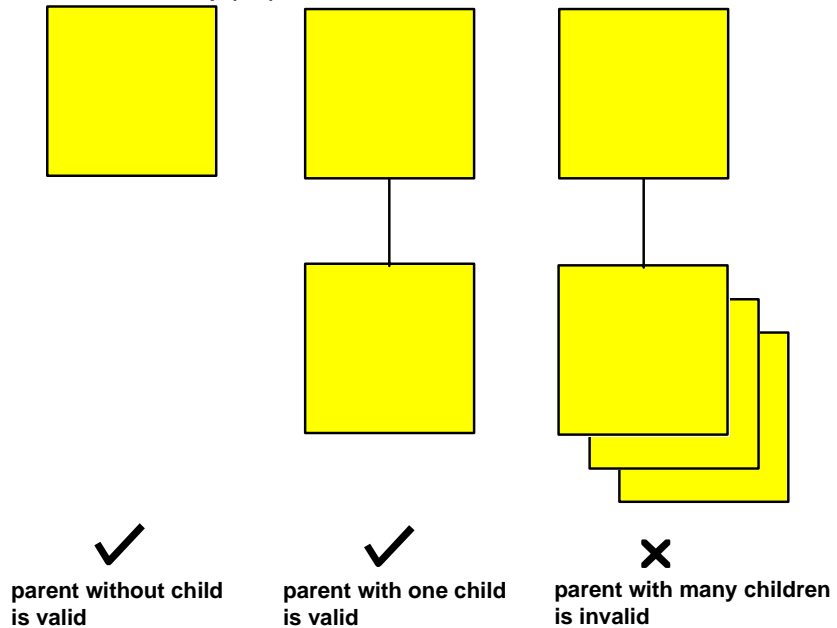
- If you are using a network data source, you can also “rotate” the data source after you have defined it, creating an alternate view that reverses some of the data relationships and enables you to access the segments in a different order.

## One-to-One Relationships

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

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

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





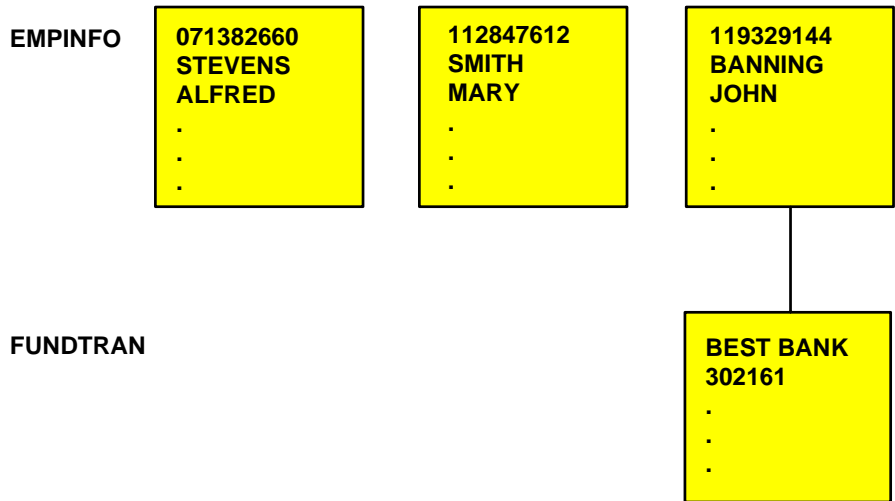
*Example*

**Understanding One-to-One Relationships**

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

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



## Where to Use One-to-One Relationships

You can specify a segment as unique to enforce a one-to-one relationship when you retrieve data.

When you retrieve data from a segment described as unique, the request treats the unique 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 One-to-One Relationships in Relational Data Sources

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

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

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

## Implementing One-to-One Relationships in FOCUS Data Sources

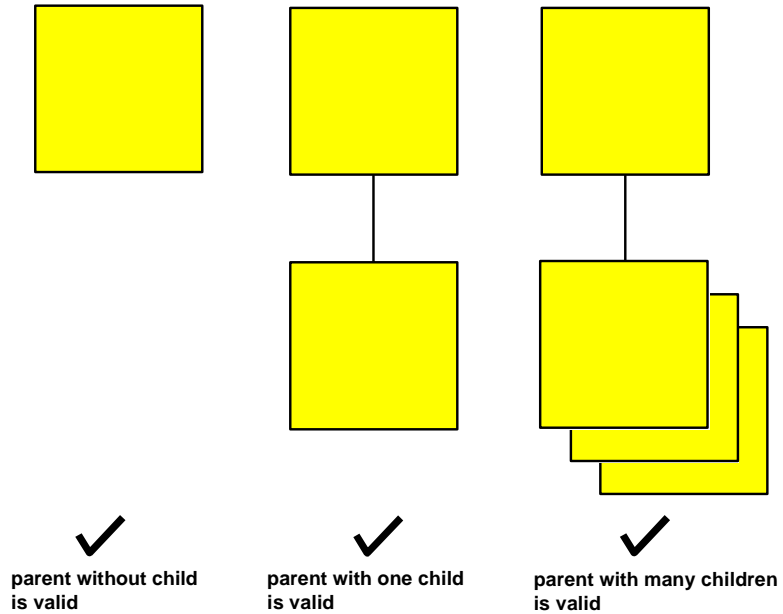
You can describe this relationship by specifying a SEGTYPE of U for the child segment. Alternately, you can join the segments by issuing the JOIN command without the ALL phrase, or by specifying a unique join in the Master File using a SEGTYPE of KU (for a static join) or DKU (for a dynamic join). All of these SEGTYPE values are described in Chapter 6, *Describing a FOCUS Data Source*.

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

# One-to-Many Relationships

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

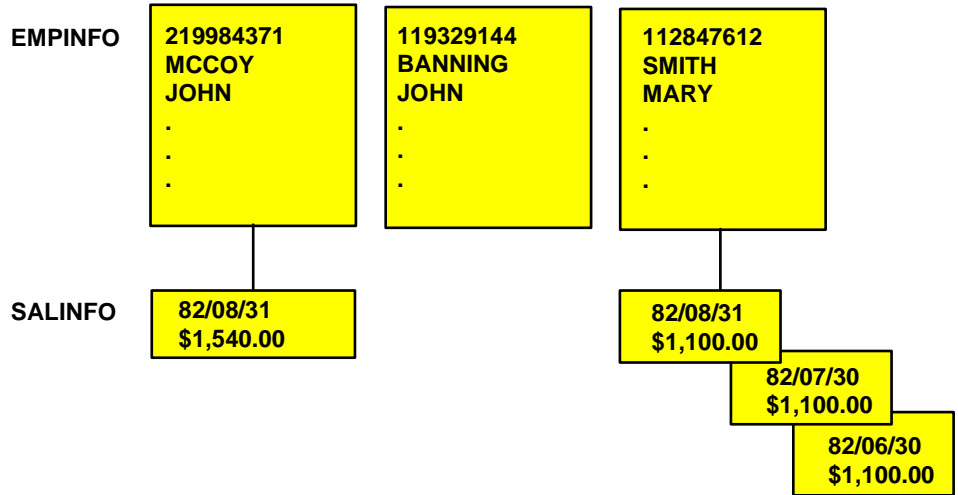
**One-to-Many Relationship (1:M)**



**Example**

**Understanding One-to-Many Relationships**

For example, in the EMPLOYEE data source, each EMPINFO segment instance describes one 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, and so the relationship between EMPINFO and SALINFO is one-to-many.



**Implementing One-to-Many Relationships in Relational Data Sources**

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

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

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*.

You can describe a one-to-many relationship between different records by using a RECTYPE field to indicate the type of each record, and the PARENT attribute to indicate the relationship between the different records. RECTYPE fields are described in Chapter 5, *Describing 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 phrase 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 One-to-Many Relationships in FOCUS Data Sources

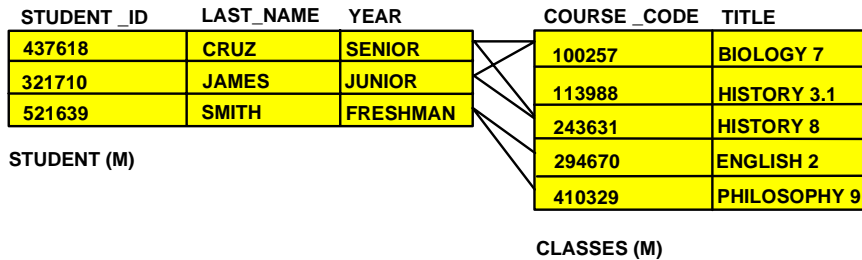
You can describe this relationship by specifying a SEGTYPE of Sn or SHn for the child segment. Alternatively, you can join the segments by issuing the JOIN command with the ALL phrase or by specifying a join in the Master File with a SEGTYPE of KM (for a static join) or DKM (for a dynamic join). All of these SEGTYPE values are described in Chapter 6, *Describing a FOCUS Data Source*.

# Many-to-Many Relationships

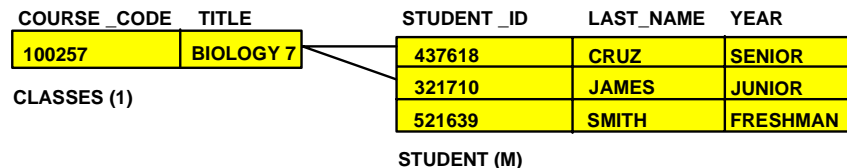
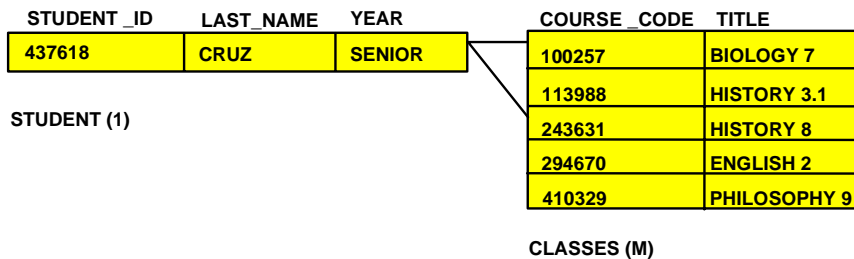
A less commonly used relationship is many-to-many; each instance of one segment can be related to one or more instances of a second segment, and each instance of the second segment can be related to one or more instances of the first segment. It is possible to implement this relationship directly between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

## Implementing Many-to-Many Directly

A direct many-to-many relationship can exist between two relational tables. For example, 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. This is illustrated in the following diagram:



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



When you 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. You describe the join in the Master File or JOIN command as you would for a standard one-to-many relationship.

*Example*

### Implementing Many-to-Many Directly

You could use the JOIN command to describe the relationship from the perspective of the STUDENT table as follows:

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

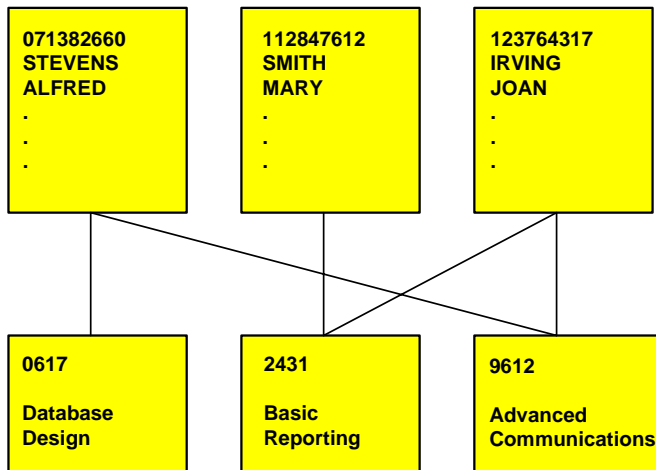
You could describe the relationship from the perspective of the CLASSES table as follows:

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

### Implementing Many-to-Many Indirectly

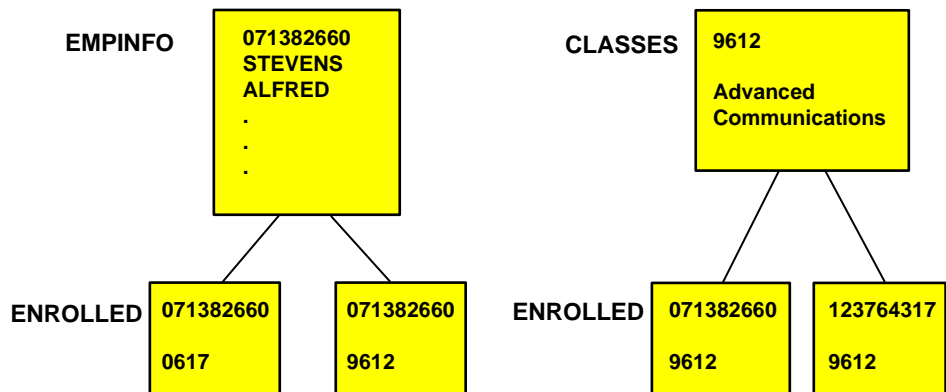
Some non-relational data sources cannot represent a many-to-many relationship directly. However, they can represent it indirectly, and you can describe it as such.

For example, 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 would describe one course. Each employee can take many courses, and many employees can take each course, so this is a many-to-many relationship.



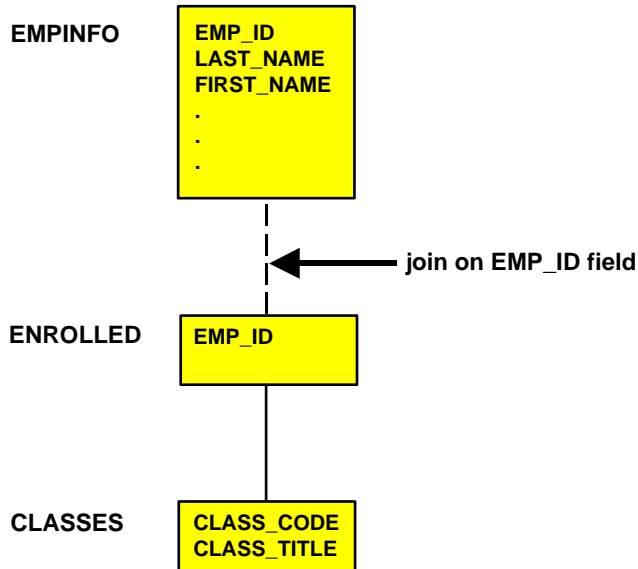
However, because some types of data sources cannot represent such a relationship directly, we need to introduce a mediating segment called ENROLLED. This new segment contains the keys from both of the original segments, EMP\_ID and CLASS\_CODE, and, in a sense, it represents the relationship between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

This is illustrated in the following diagram.





The next step is to make the mediating segment a child of one of the two original segments. For example, you can design the SCHOOL 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; now that we are making it 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:



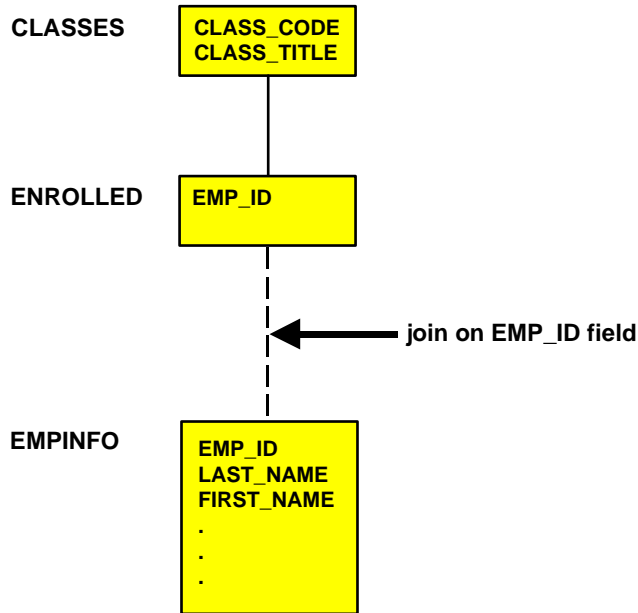
When the original M:M relationship is seen from this perspective, it looks like a 1:M:1 relationship. That is, one employee (EMPINFO) is enrolled many times (ENROLLED), and each enrollment is for a single class (CLASSES).

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. You decide which segment's perspective to use by making that segment the parent in the join. You describe the join using the JOIN command, or for FOCUS data sources, alternately in the Master File. If you make the mediating segment, in this case ENROLLED, the child (cross-referenced) segment in the join, you implement it as a standard one-to-many relationship; if you make it the parent (host), you implement it as a standard one-to-one join.

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

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

The new structure looks like the following:



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 COURSESEG.

# Recursive Relationships

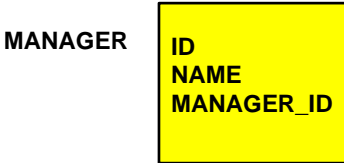
Generally, you use one-to-one and one-to-many relationships to join two different segments, usually in two different data sources. However, you can also join the same data source to itself, and even the same segment to itself. This technique, which has many useful applications, is called a recursive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

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

*Example*

## Recursive Joins With a Single Segment

For example, 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:



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

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

which would create the following structure:



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

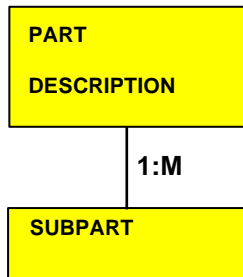
Once you have issued the join, you would be able to generate an answer set such as the following:

| 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*

### Recursive Joins With Multiple Segments

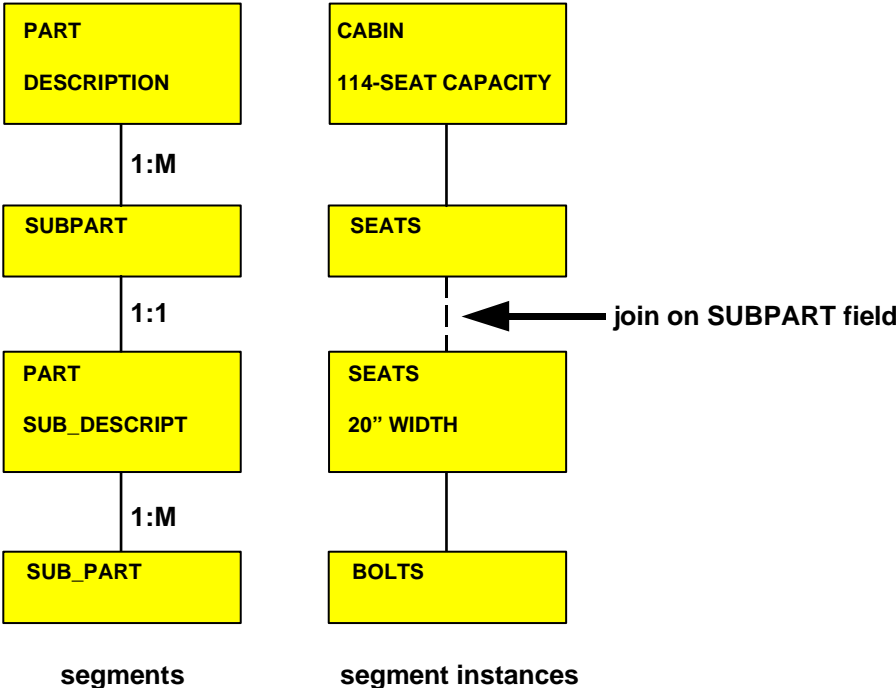
You can recursively join larger structures as well. For example, imagine a two-segment data source called AIRCRAFT that stores a bill-of-materials for an aircraft company. The root segment has the name and description of a part, and the child segment has the name of a subpart. For each part, there can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



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

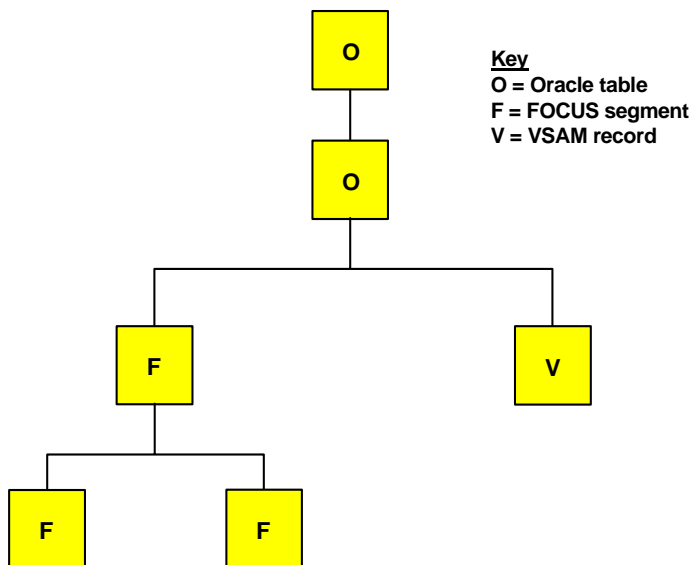
```
JOIN SUBPART IN AIRCRAFT TO PART IN AIRCRAFT AS SUB_PART
```

This produces the following data structure:



## Relating Segments From Different Types of Data Sources

The JOIN command enables you to join segments from different types of data sources, creating temporary data structures that contain related information from otherwise incompatible sources. For example, you could 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*.

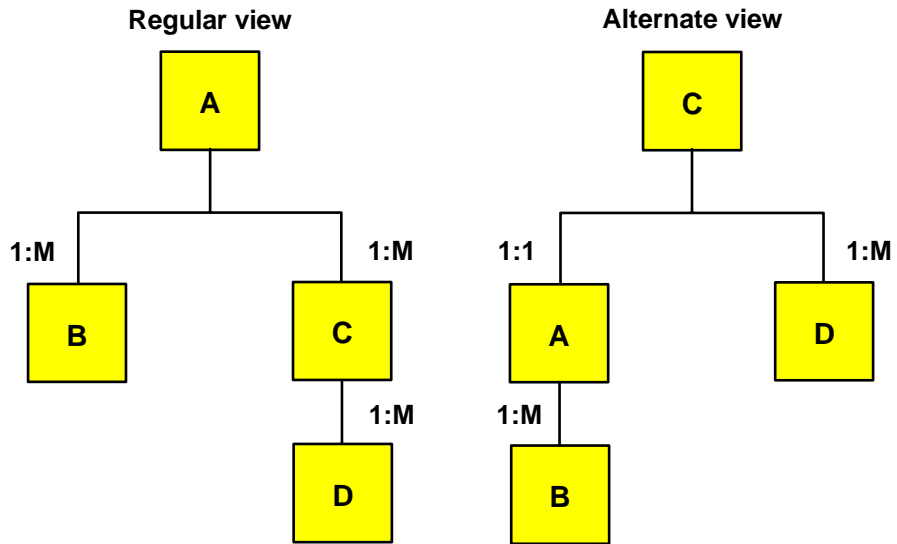
For detailed information on using the JOIN command with different types of data sources, see the *Creating Reports* manual.

## Rotating a Data Source: Alternate Views

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, creating an alternate view that changes some of the segment relationships and enables you to access the segments in a different order. By customizing the access path in this way, you can enable it to be accessed faster for a given application.

### Example

#### Rotating a Data Source



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

When you report from a data source using an alternate view, the data is accessed 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, using the WHERE or IF phrases, and make the selection criteria an equality or range test.

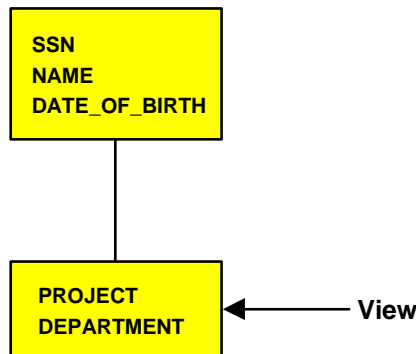
An alternate view can be requested on any segment in a data source, except a cross-referenced segment. You request an alternate view with the TABLE command by naming a field from the segment you wish to view as the new root segment. This field may not be a virtual field. The only restriction on requesting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

## Other Uses of an Alternate View

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

- 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.

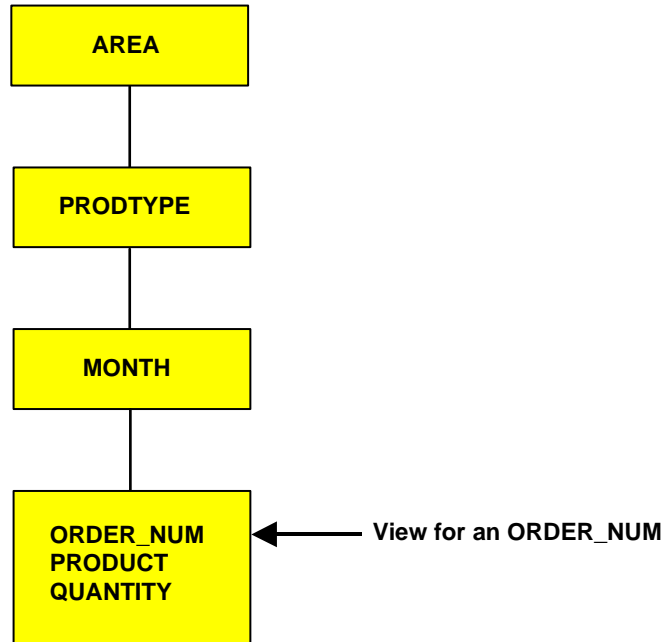
Consider a personnel system where the employee identity (SSN) is in the root segment and the department number of the project an employee is working on is in a descendant segment. Access to all employees in a given department is obtained using the view from the department.



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



- Individual record identified in descendant segments. You can use an alternate 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 often not available, then screening on it means that fewer segments have to be examined. For instance, if there are 10,000 occurrences of a parent segment, but only 2,000 of these have a given child segment, it is faster to view the data source from the vantage point of the 2,000 when the given child is involved in the screening in the request.

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

## ***Syntax***

### **How to Specify an Alternate View**

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

```
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. The field must be a real field, not a temporary field defined with the DEFINE attribute or the DEFINE or COMPUTE commands.

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

## ***Example***

### **Specifying an Alternative View**

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

```
TABLE FILE EMPLOYEE.DED_CODE
```

---

## CHAPTER 4

# Describing An Individual Field

### Topics:

- Field Characteristics
- The Field's Name: FIELDNAME
- The Field's Synonym: ALIAS
- The Displayed Data Type: USAGE
- The Stored Data Type: ACTUAL
- Null or MISSING Values: MISSING
- Validating Data: ACCEPT
- Online Help Information: HELPMESSAGE
- Alternative Report Column Titles: TITLE
- Documenting the Field: DESCRIPTION
- Describing a Virtual Field: DEFINE

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.

## Field Characteristics

The Master File describes the following field characteristics:

- The name of the field described by the `FIELDNAME` attribute.
- Another name for the field—either its original name as defined to its native data management system, or (for some types of data sources) a synonym of your own choosing, or (in some special cases) a pre-defined value that tells 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 both as it is stored in the data source and as it is formatted, since these are identical.

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

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

- The option that a field is virtual—that is, not stored in the data source—and has 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`. For detailed information, see the *Developing Applications* manual.

## The Field's Name: FIELDNAME

You 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.

Your reporting applications may influence your choice of field name. When you generate a report, each column title in the report defaults to the name of the field displayed in that column, so it will help report readers if you assign meaningful field names. Of course, you do not need to rely upon this default. You can specify a different column title within a given report by using the AS phrase in that report request—as described in the *Creating Reports* manual—or a different default column title for all reports by using the TITLE attribute in the Master File, as described in *Alternative Report Column Titles: TITLE* on page 4-49.

### *Syntax*

#### How to Identify the Field Name

`FIELD[NAME] = field_name`

where:

`field_name`

Is the name you want to use to identify 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 should begin with a letter. Other characters are not recommended and may cause problems in some operating environments or when resolving expressions.

It is recommended that you not use field names of the type Cn, En, and Xn (where n is any sequence of one or two digits) because these can be used to refer to report columns, HOLD file fields, and other special objects.

If you need to use special characters because of a field's report column title, consider using the TITLE attribute in the Master File to specify the title, as described in *Alternative Report Column Titles: TITLE* on page 4-49.

### *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. In all other situations you can change the field name.

## Using a Long and Qualified Field Name

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

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

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

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

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

*segname*

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

*fieldname*

Is the name of the field.

### 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):

- Joins

You cannot use a long name to specify a join:

- In a JOIN command, you cannot use it for a cross-referenced field in a FOCUS data source.
  - In a multi-table Master File for a relational data source, you cannot use it for the KEYFLD and IXFLD attributes in the Access File.
  - Indexed fields and text fields in FOCUS data sources cannot have field names longer than 12 characters. They can have long ALIAS names.
  - The SQL Translator supports field names up to 48 characters.
  - A field name specified in an alternate file view cannot be long or qualified.
  - CHECK FILE
- The CHECK FILE command's PICTURE and HOLD options display the first 11 characters of long names within the resulting diagram or HOLD file. A caret (>) in the 12th position indicates that the name is longer than the displayed portion.
- ?FF, ? HOLD, ? DEFINE

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

## Using 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 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, reports 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. For example:

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

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

This request prints the field with alias CARS twice.

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

- If a field name begins with characters that are the same as the name of a prefix operator, it may be unclear whether a request is referencing that field name or a second field name prefixed with the operator. 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 was between two levels of qualification. To retrieve the CAR1 field, you must specify its alias.

## The Field's Synonym: ALIAS

You can assign every field an alternative name, or alias. A field's alias may be its original name as defined to its native data source, any name of your choosing, or in special cases, a pre-defined value. The way in which you assign the alias is determined by the type of data source and, in special cases, the role the field plays in the data source. Once it has been assigned, you can use this alias in requests as a synonym for the regular field name. You 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 this—they are functionally identical, refer to the same field, and produce the same result:

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

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

**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. You assign a value to ALIAS in the following way for the following types of data sources:

- **Relational data sources.** ALIAS describes the field's original column name as defined in the relational table.
- **Sequential data 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. ALIAS is optional.

Note that ALIAS is used in a different way for sequenced repeating fields, where its value is ORDER, as well as for RECTYPE and MAPVALUE fields when the data source includes multiple record types. See Chapter 5, *Describing a Sequential, VSAM, or ISAM Data Source*, for more information about using ALIAS.

- **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. ALIAS is optional. See Chapter 6, *Describing a FOCUS Data Source*, for more information about using ALIAS. 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 you want a field to be formatted when it displays in reports or is used in calculations.

For FOCUS data sources, which do not use the ACTUAL attribute, USAGE also specifies how the field is to be stored. For other types of data sources, you will usually want to 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. 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. You can use these options to customize how the field is displayed in reports.

### Syntax

### How to Specify a Display Format

`USAGE = t1[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).

*1*

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

*d*

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

The complete 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 be displayed with the USAGE format given (for example, the result of aggregation is too large) asterisks are displayed.

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

## 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 4-16 for additional information about numeric formats.
- **Alphanumeric.**
- **Date.** The date format enables you to define date components such as year, quarter, month, day, and day of week; to sort by date; to do date comparisons and arithmetic with dates; and to automatically validate dates in transactions. Note that for some applications, such as assigning a date value using the DECODE function, you may wish to instead use alphanumeric, integer, or packed-decimal fields with date display options which provide partial date functionality.
- **Date-Time.**
- **Text.**

## Integer Format

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

You can also use integer fields with date display options to provide limited date support. This use of integer fields is described in the *Alphanumeric and Numeric Formats with Date Display Options* on page 4-32.

The integer USAGE type is I. Display options are described in *Numeric Display Options* on page 4-16. The format of the length specification is

*n*

where:

*n*

Is the maximum number of digits. The maximum integer size is 10 digits with I11 reserved for a negative sign. The maximum integer value displayed is 2147483647.

For example:

| <b>Format</b> | <b>Display</b> |
|---------------|----------------|
| I6            | 4316           |
| I2            | 22             |
| I4            | -617           |

## Floating-Point Double-Precision Format

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

The floating-point double-precision USAGE type is D. The compatible display options are described in *Numeric Display Options* on page 4-16. The length specification format is

*t[.s]*

where:

*t*

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

*s*

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

For example:

| <b>Format</b> | <b>Display</b> |
|---------------|----------------|
| 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. The compatible display options are described in *Numeric Display Options* on page 4-16. The length specification format is

*t[.s]*

where:

*t*

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

*s*

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

For example:

| <b>Format</b> | <b>Display</b> |
|---------------|----------------|
| F5.1          | 614.2          |
| F4            | 318            |

## Packed-Decimal Format

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

You can also use packed-decimal fields with date display options to provide limited date support. This use of packed-decimal fields is described in *Alphanumeric and Numeric Formats with Date Display Options* on page 4-32.

The packed-decimal USAGE type is P. The compatible display options are described in *Numeric Display Options* on page 4-16.

The length specification format is

*m.n*

where:

*m*

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

*n*

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

For example:

| <b>Format</b> | <b>Display</b> |
|---------------|----------------|
| P9.3          | 4168.368       |
| P7            | 617542         |

## Numeric Display Options

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

| Edit Option | Meaning                        | Effect                                                                                                                                                                                         |
|-------------|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| %           | 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.<br>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.                                                                                                                                                                           |
| M           | Floating \$ (for US code page) | Places a floating dollar sign \$ to the left of the highest significant digit.<br><b>Note:</b> The currency symbol displayed depends on the code page used.                                    |
| MDY         | Month-Day-Year                 | Displays alphanumeric or integer data as a date in the form month/day/year.                                                                                                                    |
| N           | Fixed \$ (for US code page)    | Places a dollar sign \$ to the left of the field. The symbol displays only on the first detail line of each page.<br><b>Note:</b> The currency symbol displayed depends on the code page used. |
| 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.                                                                                                                    |

## Extended Currency Symbol Display Options

Extended currency symbol format options allow you to display the following currency symbols regardless of the code page used: US dollar, euro, British pound, and Japanese yen. The extended currency symbol format options consist of two characters: an exclamation point followed by one of the supported upper or lower case letters. An upper case letter displays a floating currency symbol on each detail line. A lower case letter displays a fixed currency symbol to the left of the field on the first detail line of each report page. These options are valid for numeric formats (I, D, F, and P).

Use the following character combinations as the final two characters in any numeric display format:

| Display Option | Description                  | Example |
|----------------|------------------------------|---------|
| !d             | Fixed dollar sign            | D12.2!d |
| !D             | Floating dollar sign         | D12.2!D |
| !e             | Fixed euro symbol            | F10.2!e |
| !E             | Floating euro symbol         | F10.2!E |
| !l             | Fixed British pound sign     | D12.1!l |
| !L             | Floating British pound sign  | D12.1!L |
| !y             | Fixed Japanese yen symbol    | I9!y    |
| !Y             | Floating Japanese yen symbol | I9!Y    |

### Reference

#### Usage Notes for Extended Currency Symbol Support

- The extended currency option must be the last option in the format.
- The extended currency option cannot be used in the same format specification as M or N.
- In order to print the extended currency symbols, you must be sure they are supported by the fonts accessible to your printer.
- Using a fixed currency symbol places the symbol only on the first line of each report page. If you use field-based reformatting to display multiple currency symbols in one report column, only the symbol associated with the first row displays. In this case, you should use floating currency symbols.
- In TSO, when you display report output without HotScreen (SET SCREEN=OFF), by default the extended currency symbols do not display because the terminal I/O procedures translate all terminal output to characters that appear in USA EBCDIC keyboard layouts and code charts. You can change this default behavior with the SET TRANTERM = OFF command.

### Example

### Displaying Extended Currency Symbols

The following request uses field-based reformatting to display the Japanese yen on the report row that represents Japan, the British pound on the row that represents England, and the euro on the row that represents Italy. Note that the comma inclusion display option (C) in the format for England is specified prior to the currency option:

```
DEFINE FILE CAR
CFORMAT/A8 = DECODE COUNTRY('ENGLAND' 'F12.1C!L' 'JAPAN' 'D12!Y'
                          ELSE 'D12.2!E');
END

TABLE FILE CAR
PRINT SALES/CFORMAT DEALER_COST/CFORMAT
BY COUNTRY
  WHERE COUNTRY EQ 'ENGLAND' OR 'JAPAN' OR 'ITALY'
  WHERE SALES GT 0
END
```

The output is:

| COUNTRY | SALES      | DEALER_COST |
|---------|------------|-------------|
| -----   | -----      | -----       |
| ENGLAND | £12,000.0  | £11,194.0   |
| ITALY   | €30,200.00 | €16,235.00  |
| JAPAN   | ¥78,030    | ¥5,512      |

**Example****Using Numeric Display Options**

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

| Option              | Format | Data     | Display      |
|---------------------|--------|----------|--------------|
| Percent sign        | I2%    | 21       | 21%          |
|                     | D7%    | 6148     | 6,148%       |
|                     | F3.2%  | 48       | 48.00%       |
| Comma suppression   | D6c    | 41376    | 41376        |
|                     | D7Mc   | 6148     | \$6148       |
|                     | D7Nc   | 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   | 0.123456D+04 |
| Year/month/day      | I6YMD  | 980421   | 98/04/21     |
|                     | I8YYMD | 19980421 | 1998/04/21   |
| Month/day/year      | I6MDY  | 042198   | 04/21/98     |
|                     | I8MDYY | 04211998 | 04/21/1998   |
| Day/month/year      | I6DMY  | 210498   | 21/04/98     |
|                     | I8DMYY | 21041998 | 21/04/1998   |
| Month translation   | I2MT   | 07       | JUL          |

Several display options can be combined, as shown:

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

All of the options may be specified in any order. Options M and N (floating and non-floating dollar sign) and data format D (floating-point double-precision) automatically invoke option C (comma). Options L and S cannot be used together. Option T (translate month) can be included anywhere in an alphanumeric or integer USAGE specification that includes the M (month) display option. Date display options (D, M, T, and Y), which cannot be used with floating-point fields, are described in *Alphanumeric and Numeric Formats with Date Display Options* on page 4-32.

## Alphanumeric Format

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

You can also use alphanumeric fields with date display options to provide limited date support. This use of alphanumeric fields is described in *Alphanumeric and Numeric Formats with Date Display Options* on page 4-32.

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 or FUSION 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:

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

The standard numeric display options are not available for the alphanumeric data format. However, alphanumeric data can be printed under the control of a pattern that is supplied at run time. For instance, if a product code is to be displayed in parts, with each part separated by a "-", the following could be included 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, PRODCODE will be displayed 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 and FUSION data sources, a segment still has to fit on a 4K page. Thus, the maximum length of an alphanumeric field is dependent 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 are unable to 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 and manipulate the field's value and display that value in ways appropriate to a date. Using date format, you can:

- Define date components such as year, quarter, month, day, and day of week, and extract them easily from date fields.
- Sort reports into date sequence, regardless of how the date is displayed.
- 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

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.                                                                                                                                                                            |
| t              | Translate month or day | Functions the same as uppercase T (described above), except that the first letter of the month or day is uppercase and the following letters are lowercase.*                                                                                                                        |
| TR             | Translate month or day | Functions the same as uppercase T (described above), except that the entire month or day name is printed instead of an abbreviation.                                                                                                                                                |
| tr             | Translate month or day | Functions the same as lowercase t (described above), except that the entire month or day name is printed instead of an abbreviation.*                                                                                                                                               |
| Q              | Quarter                | Prints the quarter (1 - 4 if Q is specified by itself, or Q1 - Q4 if it is specified together with other date format items such as Y).                                                                                                                                              |
| W              | Day-of-Week            | If it is included in a 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.*                                                                                                                                            |



| Display Option     | Meaning       | Effect                                                                                                                                                                                                       |
|--------------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>WR</code>    | Day-of-Week   | Functions the same as uppercase W (described above), except that the entire day name is printed instead of an abbreviation.*                                                                                 |
| <code>wr</code>    | Day-of-Week   | Functions the same as lowercase w (described above), except that the entire day name is printed instead of an abbreviation.*                                                                                 |
| <code>JUL</code>   | Julian format | Prints date in Julian format.                                                                                                                                                                                |
| <code>YYJUL</code> | 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 will not sort based on DEFCENT and YRTHRESH settings. A field with a format of Y will not equal a YY field, as this is not a displacement, but a 4-digit integer.
- It is possible to use DEFCENT and YRTHRESH to convert a field from Y to YY format.

**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.

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

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

**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.

## Controlling the Date Separator

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

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

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

## Date Translation

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

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

## Example Using a Date Format

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

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

Note that the date attributes in the ACTUAL format specify the order in which the date is stored in the non-FOCUS data source. If the ACTUAL format does not specify the order of the month, day, and year, it will be inferred from the USAGE format.

## Using a Date Field

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

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

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

Natural date literals can be used in all date computations and all methods of data source updating. Examples are shown in the following chart.

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

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

The following chart describes the format of natural date literals.

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

The date components of a natural date literal can be specified in any order, regardless of their order in the 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 would both be valid literals.

## Numeric Date Literals

Numeric date literals differ from natural date literals in that they are simple strings of digits. The order of the date components in a numeric date literal must match the order of the date components in the corresponding 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. Valid date components are Y or YY, Q, M, W, and D.

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

For example, if NEWQUARTER and THISQUARTER both have 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

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 would be 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 elapsed time since the date format base date. For each field, the unit of elapsed time is that field's smallest date component.

For example, if the 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 entered the numeric literal for February 13, 1964 (that is, 021364), and then printed the field in a report, 02/13/64 would be displayed. If you used it in the equation

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

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

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

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

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



## Displaying a Non-Standard Date Format

By default, if a date field in a non-FOCUS data source contains an invalid date, a message displays and the entire record fails to display 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 will not display. The SET ALLOWCVTERR command enables you to display the rest of the record that contains the incorrect date.

### Syntax

### How to Display a Non-Standard Date

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

where:

ON

Allows the display of 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. This 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 is displayed. 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:

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

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

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

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

## Date-Time Formats

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.

See the *Developing Applications* manual for information on subroutines for manipulating date-time fields.

## Describing a Date-Time Field

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

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

The 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 Date-Time Field

The USAGE attribute can be one of the following:

USAGE = *Hnn*

USAGE = *Htimefmt1*

USAGE = *Hdatefmt [separator] [timefmt2]*

where:

*Hnn*

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

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

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

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

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

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

*Htimefmt1*

Is the USAGE format for displaying time only. Hour, minute, and second components are always displayed separated by colons (:), with no intervening blanks. Unless you specify one of the AM/PM time display options, the time component is displayed using the 24-hour system.

When the format includes more than one time display option:

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

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

| Option | Meaning                                                                                                                                                                       | Effect                                                                                                                                          |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| H      | hour (two digits)<br>If the format includes the option a or A, the hour value is from 01 to 12.<br>Otherwise, the hour value is from 00 to 23, with 00 representing midnight. | Prints a two-digit hour. For example:<br><code>USAGE = HH prints 02</code>                                                                      |
| h      | hour with zero suppression<br>If the format includes the option a or A, the hour value is from 1 to 12.<br>Otherwise, the hour is from 0 to 23.                               | Displays the hour with zero suppression. For example:<br><code>USAGE = Hh prints 2</code>                                                       |
| I      | minute (two digits)<br>The minute value is from 00 to 59.                                                                                                                     | Prints the two-digit minute. For example:<br><code>USAGE = HHI prints 02:05</code>                                                              |
| i      | minute with zero suppression<br>The minute value is from 0 to 59.                                                                                                             | Prints the minute with zero suppression. Cannot be used together with an hour format (H or h). For example:<br><code>USAGE = Hi prints 5</code> |
| S      | Second (two digits)<br>00 to 59                                                                                                                                               | Prints the two-digit second. For example:<br><code>USAGE = HHIS prints 02:05:27</code>                                                          |

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

*Hdatefmt*

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

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

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

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

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

*separator*

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

*timefmt2*

Is the format for a time that follows a date. Time is separated from the date by a blank; time components are separated from each other by colons. Unlike the format for time alone, a time format that follows a date format consists of at most two characters: a single character to represent all of the time components to be displayed and, optionally, one character for an AM/PM option.

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

| Option | Meaning                                                                                           | Example                                             |
|--------|---------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| H      | Prints hour                                                                                       | USAGE = HYYMDH prints<br>1999/02/05 02              |
| I      | Prints hour:minute                                                                                | USAGE = HYYMDI prints<br>1999/02/05 02:05           |
| S      | Prints hour:minute:second                                                                         | USAGE = HYYMDS prints<br>1999/02/05 02:05:25        |
| s      | Prints hour:minute:second.millisecond                                                             | USAGE = HYYMDSs prints<br>1999/02/05 02:05:25.444   |
| m      | Prints hour:minute:second.microsecond                                                             | USAGE = HYYMDm prints<br>1999/02/05 02:05:25.444555 |
| A      | Prints AM or PM. Uses the 12-hour system and causes the hour to be printed with zero suppression. | USAGE = HYYMDSA prints<br>1999/02/05 2:05:25AM      |
| a      | Prints am or pm. Uses the 12-hour system and causes the hour to be printed with zero suppression. | USAGE = HYYMDSa prints<br>1999/02/05 2:05:25am      |

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

## Specifying a Date-Time Value

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

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

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

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

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

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

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

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

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

where:

*time\_string*

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

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

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

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



*date\_string*

Can have one of the following three formats:

- **Numeric string format** is exactly four, six, or eight digits. Four-digit strings are considered to be a year (century must be specified); the month and day are set to January 1. Six and eight-digit strings contain two or four digits for the year, followed by two for the month, and then two for the day. Because the component order is fixed with this format, the DATEFORMAT setting described in the *Developing Applications* manual is ignored.

If a numeric-string format longer than eight digits is encountered, it is treated as a combined date-time string in the *Hnn* format described in *Date-Time Formats* on page 4-32. The following are examples of numeric string date constants:

```
99
1999
19990201
```

- **Formatted-string format** contains a one or two-digit day, a one or two-digit month, and a two or four-digit year separated by spaces, slashes, hyphens, or periods. All three parts must be present and follow the DATEFORMAT setting described in the *Developing Applications* manual. If any of the three fields is four digits, it is interpreted as the year, and the other two fields must follow the order given by the DATEFORMAT setting. The following are examples of formatted-string date constants:

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

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

```
January 6 2000
```

**Note:**

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

## Text Field Format

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

where:

`fieldname`

Is the name you assign the text field.

`aliasname`

Is an alternate name for the field name.

`nn`

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

`F`

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

For example, the text field in the COURSES 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.

| <b>Format</b> | <b>Display</b>                                                                                                           |
|---------------|--------------------------------------------------------------------------------------------------------------------------|
| 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.

## The Stored Data Type: ACTUAL

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

### The ACTUAL Attribute

This attribute describes the type and length of your data as it actually exists in the 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 tables.

The following table 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.                                                                                                      |
| An          | 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.                         |
| D8          | Double-precision, floating-point numbers, stored internally in eight bytes.                                                                                                                                                       |
| F4          | Single-precision, floating-point numbers, stored internally in four bytes.                                                                                                                                                        |
| In          | Binary integers:<br>I1 = single-byte binary integer.<br>I2 = half-word binary integer (2 bytes).<br>I4 = full-word binary integer (4 bytes).                                                                                      |
| Pn          | Where $n = 1-16$ . Packed decimal internal format. $n$ is number of bytes, each of which contains two digits, except for the last byte which contains a digit and the sign (+ or -). For example, P6 means 11 digits plus a sign. |

| ACTUAL Type | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Zn          | Where $n = 1-31$ . Zoned decimal internal format. $n$ is the number of digits, each of which takes a byte of storage. The last digit contains a digit and the sign.<br>If the field contains an assumed decimal point, represent the field with an ACTUAL format of Zn and a USAGE format of Pm.d, where $m$ is the total number of digits in the display plus the assumed decimal point, and $d$ is the number of decimal places. $m$ must be at least 1 greater than the value of $n$ . For example, a field with ACTUAL=Z5 and one decimal place would need USAGE=P6.1 (or P7.1, or greater). |

**Note:**

- Unless your data source was created by a program, all of the characters will be characters of either type A (alphanumeric) or type Z (zoned decimal).
- The ASQ. prefix is not valid for a packed field of any length.

*Reference*

**ACTUAL to USAGE Conversion**

The following conversions from ACTUAL format to USAGE (display) format are permitted:

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

*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                     | X(4)                   | 4                | A4            | A4           |
| DISPLAY                     | S99                    | 2                | Z2            | P3           |
| DISPLAY                     | 9(5)V9                 | 6                | Z6.1          | P8.1         |
| DISPLAY                     | 99                     | 2                | A2            | A2           |
| COMP                        | S9                     | 4                | I2            | I1           |
| COMP                        | S9(4)                  | 4                | I2            | I4           |
| COMP*                       | S9(5)                  | 4                | I4            | I5           |
| COMP                        | S9(9)                  | 4                | I4            | I9           |
| COMP-1**                    | —                      | 4                | F4            | F6           |
| COMP-2***                   | —                      | 8                | D8            | D15          |
| COMP-3                      | 9                      | 8                | P1            | P1           |
| COMP-3                      | S9V99                  | 8                | P2            | P5.2         |
| COMP-3                      | 9(4)V9(3)              | 8                | P4            | P8.3         |
| FIXED BINARY(7)<br>(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:**

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

## Null or MISSING Values: MISSING

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

If your type of data source supports missing data, as do FOCUS 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 set to ON, 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 be displayed, a special character is displayed 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, 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. This is the default value.

## Reference

### Usage Notes for MISSING

Note the following rules when using MISSING:

- **Alias.** MISSING does not have an alias.
- **Setting.** It is recommended that you set the MISSING attribute to match the field's predefined null characteristic (whether the characteristic was explicitly set when the data source was created, or set by default). For example, if a relational table column has been created with the ability to accept null data, you should describe the field with the MISSING attribute set to ON so that its null values are correctly interpreted. This is not a consideration for FOCUS data sources, for which the field declaration in the Master File both defines the field and describes it.
- **Changes.** You can change the MISSING attribute at any time. Note that changing MISSING will not affect the actual stored data values that had been entered using the old setting. However, it will affect how that data is interpreted: if null data is entered when MISSING is set to ON, and then MISSING is switched to OFF, the data originally entered as null will be interpreted as blanks (for alphanumeric fields) or zeroes (for numeric fields). The only exception is FOCUS data sources, in which the data originally entered as missing will be 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 will automatically be supplied for the two blank records. Thus, the average of these four records will be  $(0+0+1+3)/4$  or 1. If you set MISSING to ON, the two blank records will not be used in the calculation, so the average will be  $(1+3)/2$  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 others is that they are not stored. You do have to supply a missing attribute for fields in unique segments on which you want to perform counts or averages.

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

## Validating Data: ACCEPT

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

**Note:** Suffix VSAM and FIX 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 = range  
ACCEPT = FIND (field [AS name] IN file)
```

where:

*list*

Is a string of acceptable values. The syntax is:

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

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

*range*

Gives the range of acceptable values. The syntax is:

```
value1 TO value2
```

For example, ACCEPT = 150 TO 1000.

FIND

Verifies the incoming data against the values in another indexed field. This option is available only for FOCUS data sources. See Chapter 6, *Describing a FOCUS Data Source*, for more information.

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

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

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



## Reference

### Usage Notes for ACCEPT

Note the following 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 the ACCEPT attribute to be propagated into the Master File of a HOLD file, use the SET HOLDATTR command. HOLD files are discussed in the *Creating Reports* manual.
- **ACCEPT** is used only in MODIFY procedures. It is useful for providing one central validation list to be used by several procedures. The FIND function is useful when the list of values is large or undergoes frequent change.
- **The HELPMESSAGE attribute** defines a message to display based on the results of an ACCEPT test.

## Online Help Information: HELPMESSAGE

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

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

## 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 is displayed. Then the specified HELPMESSAGE text is displayed:

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

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

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

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

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

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

## Setting a HELP (PF) Key

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

```
SET PFnn = HELP
```

where:

*nn*

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

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

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

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

## Alternative Report Column Titles: TITLE

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

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

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

### Syntax

#### How to Specify 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 (.). You can include blanks at the end of a column title by including a slash (/) in the final blank position. You must enclose the string within single quotation marks 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 TITLE=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.** If you wish the TITLE attribute to be propagated into the Master File of a HOLD file, use the SET HOLDATTR command. HOLD files are discussed in the *Creating Reports* manual.

## Documenting the Field: DESCRIPTION

DESCRIPTION is an optional attribute that enables you to provide comments and other documentation for a field within the Master File. You can include any comment up to 78 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.

### 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 78 characters. If the string 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.** If you use the DESCRIPTION attribute for a virtual field created with the DEFINE attribute, the DESCRIPTION attribute must be on the same line as the semicolon (;) terminating the DEFINE expression or, if there are other attributes in the declaration (such as TITLE), on the last line of the declaration.

## Describing a Virtual Field: DEFINE

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 = 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 4-3.

##### *format*

Is the field's format. The format is specified in the same way as formats assigned using the USAGE attribute, which is described in *The Displayed Data Type: USAGE* on page 4-12. 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. Any additional attributes must be on the same line as the semicolon that ends the DEFINE expression. An exception is made for the DESCRIPTION attribute. However, if you put it on the final line of the declaration, it does not need to be on the same line as the semicolon.

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

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

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

## Reference

### Usage Notes for 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.

## Using a Virtual Field

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

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

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

A DEFINE attribute in a Master File can refer to only fields in its own path. If you want to create a virtual field that derives its value from fields in several different paths, you will 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 will be used only 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.

**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.
```

---

## CHAPTER 5

# Describing a Sequential, VSAM, or ISAM Data Source

### Topics:

- Sequential Data Source Formats
- Standard Master File Attributes for a Sequential Data Source
- Standard Master File Attributes for a VSAM or ISAM Data Source
- Describing a Multiply Occurring Field in a Free-Format Data Source
- Describing a Multiply Occurring Field in a Fixed-Format, VSAM, or ISAM Data Source
- Redefining a Field in a Non-FOCUS Data Source
- Extra-Large Record Length Support
- Describing Multiple Record Types
- Combining Multiply Occurring Fields and Multiple Record Types
- Establishing VSAM Data and Index Buffers
- Using a VSAM Alternate Index
- Describing a Token-Delimited Data Source
- Reading a Complex Data Source With a User-Written Procedure

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 were 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.

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

Only ESDS and KSDS data sources are supported. If you wish to retrieve data from RRDS VSAM data sources, you may code your own access routine using SUFFIX=PRIVATE. See Appendix C, *User Exits for a Non-FOCUS Data Source* for details. For information about updating VSAM data sources, see the *FOCUS for S/390 VSAM Write Data Adapter User's Manual*.

## Sequential Data Source Formats

Sequential data sources formatted in the following ways are recognized:

- Fixed-format, in which each field occupies a pre-defined 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.

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 5-45.

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 pre-defined position in the record. You describe the record format in the Master File.

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

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

The simplest form of a fixed-record 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 can 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 (,\$). The LRECL for a Master File is 80. The RECFM for a Master File is F.
- 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 need to describe only those fields to which you intend to refer. This is very significant when using existing data sources, because they frequently contain information that you do not need for your requests. You describe only the fields you wish to include in your reports or calculations and use filler fields to represent the rest of the logical record length (LRECL) of the 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,$
```

We recommend including 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?

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

Note that comma-delimited and tab-delimited data sources cannot participate in joins.

### *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:

- One style is called free-format and is described in *What Is a Free-Format Data Source?* on page 5-5. Character values are not enclosed in double quotation marks, and the comma-dollar sign character combination is the record terminator. 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 crlf (carriage-return, line-feed) character combination is the record terminator. 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 correctly interpret the double quotation marks around character values. 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 will be 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 crlf 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 will be 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 crlf 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 as a record terminator.

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

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

The order in which the data values are described in the Master File plays an important role in comma-delimited 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 will contain the same number of commas used as delimiters as there are data fields. It will also have a dollar sign following the last comma, signaling the end of the logical record. Thus, a logical record containing ten data fields will contain ten commas as delimiters, plus a dollar sign record terminator.

A logical record may occupy as many lines in the data 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 defaults to 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 will not join. However, they are recommended 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

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

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, for instance, 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 is returned:

(FOC1016) INVALID KEY DESCRIPTION IN MASTER FILE

The cluster key definition is compared to the Master File for length and displacement. When you expand on the key in a RECTYPE 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. 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 or P16 (sign and decimal for a total of 15 digits). Fields that are 16 bytes will have a USAGE of P17 or larger.
- Fields of type D have a value of 8.
- Alphanumeric fields have a value equal to the number of characters they contain as their field length.

**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.

## Syntax

### How to Describe a VSAM Group Field

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

where:

*keyname*

Can have up to 66 characters.

## Example

### Describing a VSAM Group Field

In the library data source, the first field, PUBNO, could 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 were a VSAM data structure. The Master File would look as follows:

```
FILE = LIBRARY5, SUFFIX = VSAM, $
SEGMENT = ROOT, SEGTYPE = SO, $
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 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 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 it had the last time a value 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 will carry over for each descendant segment.

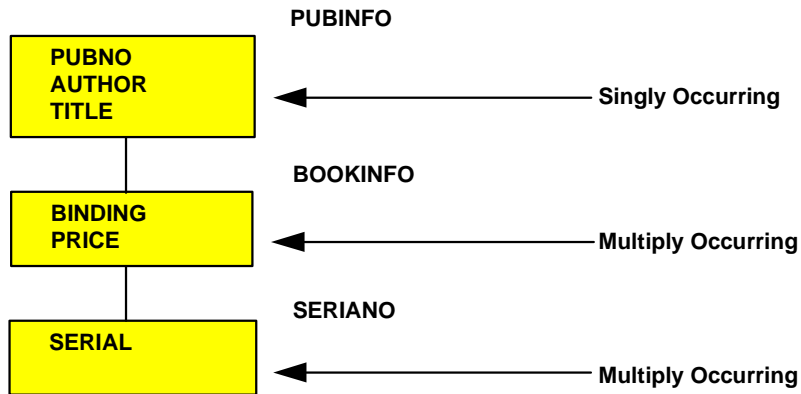
**Example**

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

Consider our 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:



You describe this data source with the following data source description:

```
FILENAME = LIBRARY4, SUFFIX = COM, $
SEGNAME = PUBINFO, SEGTYPE=S0, $
  FIELDNAME = PUBNO, ALIAS = PN, USAGE = A10, ACTUAL = A10, $
  FIELDNAME = AUTHOR, ALIAS = AT, USAGE = A25, ACTUAL = A25, $
  FIELDNAME = TITLE, ALIAS = TL, USAGE = A50, ACTUAL = A50, $
SEGNAME = BOOKINFO, PARENT = PUBINFO, SEGTYPE=S0, $
  FIELDNAME = BINDING, ALIAS = BI, USAGE = A1, ACTUAL = A1, $
  FIELDNAME = PRICE, ALIAS = PR, USAGE = D8.2N, ACTUAL = D8, $
SEGNAME = SERIANO, PARENT = BOOKINFO, SEGTYPE=S0, $
  FIELDNAME = SERIAL, ALIAS = SN, USAGE = A15, ACTUAL = A15, $
```

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



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

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 will occur that number of times.

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

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

## Using the OCCURS Attribute

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

You can have several sets of repeating fields in your data structure. You describe each of these sets of fields as a separate segment in your 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 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 (that is, if the field lengths for the segment add up to 40, and 120 characters are read in, it means there are three occurrences).

You 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.

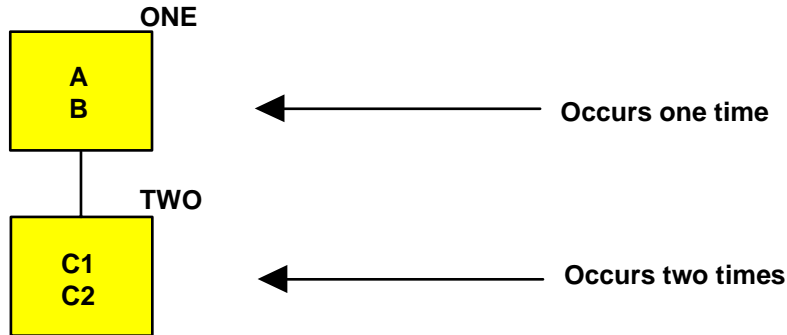
**Example**

**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:



You describe this data source with the following data source description:

```

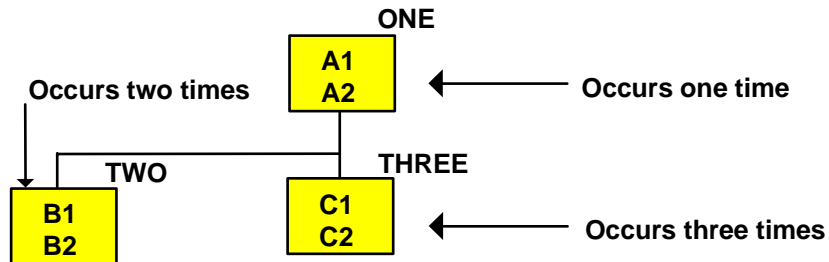
FILENAME = EXAMPLE1, SUFFIX = FIX, $
SEGNAME = ONE, SEGTYPE=S0, $
  FIELDNAME = A,      ALIAS= ,      USAGE = A2,  ACTUAL = A2, $
  FIELDNAME = B,      ALIAS= ,      USAGE = A1,  ACTUAL = A1, $
SEGNAME = TWO, PARENT = ONE, OCCURS = 2, SEGTYPE=S0, $
  FIELDNAME = C1,     ALIAS= ,      USAGE = I4,  ACTUAL = I2, $
  FIELDNAME = C2,     ALIAS= ,      USAGE = I4,  ACTUAL = I2, $
    
```

## 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:

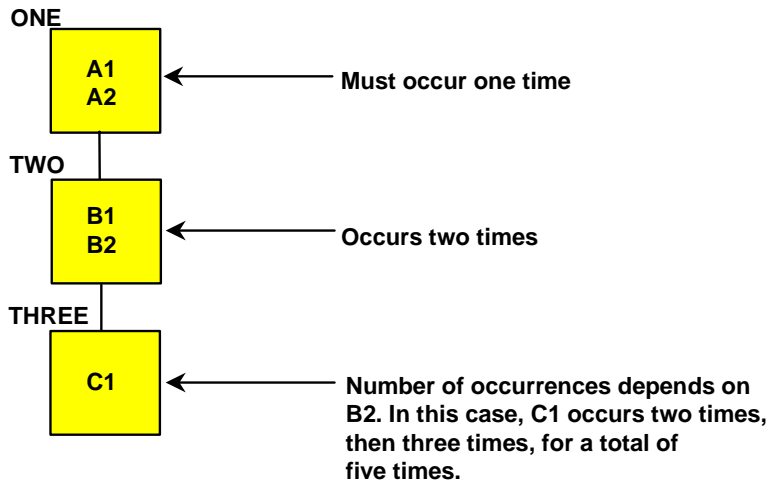


## Describing a Nested Set of Repeating Fields

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

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

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



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

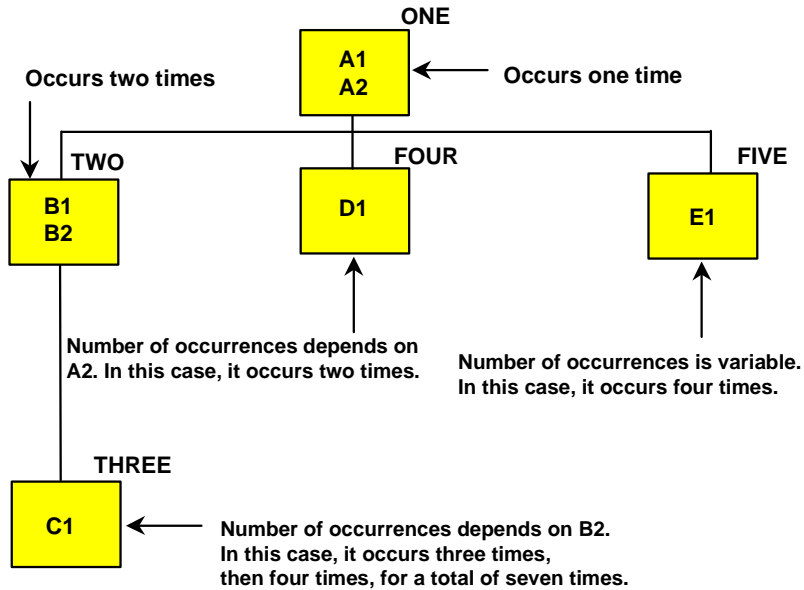
**Example**

**Describing Parallel and Nested Repeating Fields**

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

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

It produces the following data structure:



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

```

FILENAME = EXAMPLE3, SUFFIX = FIX,$
SEGNAME = ONE, SEGTYPE=S0,$
FIELDNAME = A1 ,ALIAS= ,ACTUAL = A1 ,USAGE = A1 ,,$
FIELDNAME = A2 ,ALIAS= ,ACTUAL = I1 ,USAGE = I1 ,,$
SEGNAME = TWO, SEGTYPE=S0, PARENT = ONE, OCCURS = 2 ,,$
FIELDNAME = B1 ,ALIAS= ,ACTUAL = A15 ,USAGE = A15 ,,$
FIELDNAME = B2 ,ALIAS= ,ACTUAL = I1 ,USAGE = I1 ,,$
SEGNAME = THREE, SEGTYPE=S0, PARENT = TWO, OCCURS = B2 ,,$
FIELDNAME = C1 ,ALIAS= ,ACTUAL = A25 ,USAGE = A25 ,,$
SEGNAME = FOUR, SEGTYPE=S0, PARENT = ONE, OCCURS = A2 ,,$
FIELDNAME = D1 ,ALIAS= ,ACTUAL = A15 ,USAGE = A15 ,,$
SEGNAME = FIVE, SEGTYPE=S0, PARENT = ONE, OCCURS = VARIABLE,$
FIELDNAME = E1 ,ALIAS= ,ACTUAL = A5 ,USAGE = A5 ,,$
    
```

**Note:**

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

## Using the POSITION Attribute

The POSITION attribute is an optional attribute used to describe a structure in which multiply occurring fields with an established number of occurrences are located in the middle of the record. You describe the data 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 will be 32 characters.

### *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 and overall length of the multiple field occurrences.

**Example**

**Specifying the Position of a Repeating Field**

Consider the following data structure:

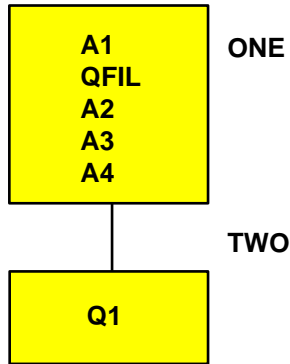
|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| A1 | Q1 | Q1 | Q1 | Q1 | A2 | A3 | A4 |
|----|----|----|----|----|----|----|----|

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 descendant, multiply occurring 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.

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

```
FILENAME = EXAMPLE3, SUFFIX = FIX,$
SEGNAME = ONE, SEGTYPE=S0,$
  FIELDNAME = A1 ,ALIAS= ,USAGE = A14 ,ACTUAL = A14 ,,$
  FIELDNAME = QFIL ,ALIAS= ,USAGE = A32 ,ACTUAL = A32 ,,$
  FIELDNAME = A2 ,ALIAS= ,USAGE = I2 ,ACTUAL = I2 ,,$
  FIELDNAME = A3 ,ALIAS= ,USAGE = A10 ,ACTUAL = A10 ,,$
  FIELDNAME = A4 ,ALIAS= ,USAGE = A15 ,ACTUAL = A15 ,,$
SEGNAME = TWO, SEGTYPE=S0, PARENT = ONE, POSITION = QFIL, OCCURS = 4 ,,$
  FIELDNAME = Q1 ,ALIAS= ,USAGE = D8 ,ACTUAL = D8 ,,$
```

This produces the following structure:



If the total length of the multiply occurring 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 will work 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 will not work.



## Specifying the ORDER Field

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

If you wish to 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 In, with any appropriate edit options.
- The ACTUAL is I4.

For example:

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

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

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

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

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

## Redefining a Field in a Non-FOCUS Data Source

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

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

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

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

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

### Example

#### Redefining a VSAM Structure

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

```
01 ALLFIELDS  
  02 FLD1  PIC X(4)           - this describes alpha/numeric data  
  02 FLD2  PIC X(4)           - this describes numeric data  
  02 RFLD1 PIC 9(5)V99 COMP-3 REDEFINES FLD2  
  02 FLD3  PIC X(8)           - this describes alpha/numeric data  
  
FILE = REDEF, SUFFIX = VSAM, $  
SEGNAME = ONE, SEGTYPE = S0, $  
  GROUP = RKEY, ALIAS = KEY, USAGE = A4 ,ACTUAL = A4 , $  
  FIELDNAME = FLD1, , USAGE = A4 ,ACTUAL = A4 , $  
  FIELDNAME = FLD2, , USAGE = A4 ,ACTUAL = A4 , $  
  FIELDNAME = FLD3, , USAGE = A8 ,ACTUAL = A8 , $  
SEGNAME = TWO, SEGTYPE = U, POSITION = FLD2, OCCURS = 1, PARENT = ONE , $  
  FIELDNAME = RFLD1, , USAGE = P8.2 ,ACTUAL = Z4 , $
```

**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.
- A field that is being redefined must be equal in length to the field that it is redefining (same actual length).
- For non-alphanumeric fields, you must know your data. Attempts to print numeric fields that contain alphanumeric data will 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

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.

To define the maximum record length, type:

```
SET MAXLRECL = nnnn
```

where *nnnn* is up to 32768.

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

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

## Describing Multiple Record Types

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 will 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 record 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. You 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 shares some other identifying piece of information, such as the PUBNO in our example. This is the field that should be included in the parent key, as well as all of its descendants' keys, to relate them.

When using the RECTYPE attribute in 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 the *Describing VSAM Positionally Related Records* example on page 5-26, the RECTYPE field is added to the group key in the SERIANO segment, the lowest descendant segment in the chain.

## Describing a 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 could contain a different two-digit code for each unique record type. You describe this identifying field with the field name RECTYPE.

Another technique for redefining parts of records is to use the MAPFIELD and MAPVALUE attributes described in *Describing a Repeating Group Using MAPFIELD* on page 5-38.

### 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 will be ignored. The syntax to describe the RECTYPE field is

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

where:

*value*

Is the record type in alphanumeric format.

*format*

Is the data type of the field. 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).

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 (+ or -). For example, P6 means 11 digits plus a sign.

If the field contains an assumed decimal point, represent the field with a USAGE format of *Pm.n*, where *m* is the total number of digits, and *n* is the number of decimal places. Thus P11.1 means an eleven-digit number with one decimal place.

### Example Specifying the RECTYPE Field

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

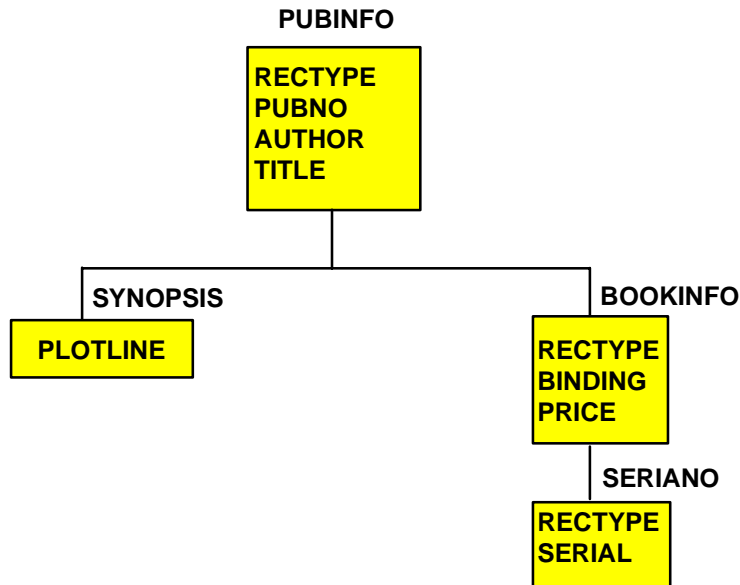
```
FIELD = RECTYPE, ALIAS = 1, USAGE = P1, ACTUAL = P1, $
```

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

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

## Describing Positionally Related Records

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



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

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

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

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

You would use the following Master File to describe this data 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 the OCCURS segment, contains a field named RECTYPE and that the ALIAS for the field contains a unique value for each segment (P, B, and S). If a record in this data source is encountered with a RECTYPE other than P, B, or S, the record will be ignored. The RECTYPE field must fall in the same physical location in each record.

## Ordering of Records in the Data Source

With sequential records, physical order determines parent/child relationships. Every parent record need not have descendants. You can specify how you want data in missing segment instances handled in your reports by using the SET command to change the ALL parameter. The SET command is described in the *Developing Applications* manual.

In the structure shown in the example in *Describing Positionally Related Records* on page 5-24, 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 will appear in the PUBINFO record. The next record may be a BOOKINFO or even another PUBINFO (in which case the first PUBINFO is assumed to have no descendants). Any SERIANO records are assumed to be descendants of the previous BOOKINFO record. If a SERIANO record follows a PUBINFO record with no intervening BOOKINFO, it is treated as if it has no parent.

**Example**

**Describing VSAM Positionally Related Records**

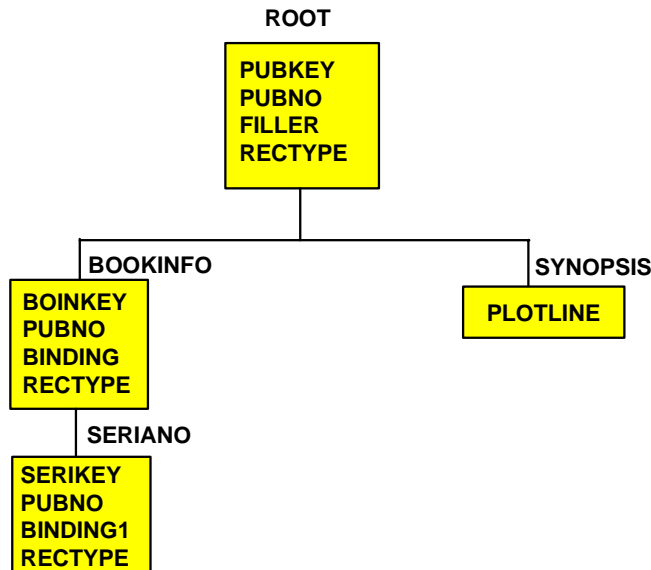
Consider the following VSAM data source that contains three types of records. The ROOT records have a key that consists of the publisher's number, PUBNO. The BOOKINFO segment has a key that consists of that same publisher's number, plus a hard- or soft-cover indicator, BINDING. The SERIANO segment key 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=POTL  ,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.



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 might request information on price and call numbers for a specific publisher's number:

```
PRINT PRICE AND SERIAL BY PUBNO
IF PUBNO EQ 1234567890 OR 9876054321
```

Since PUBNO is part of the key, the retrieval can be made quickly and the processing continues. To further speed retrieval you could add search criteria based on the BINDING field, which is also part of the key.

## Describing 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, you define a dummy root segment. You make the record types descendants of a dummy root segment. The following rules apply to the dummy root segment:

- The name of the root segment must be DUMMY.
- It must have only one field with an empty name and alias.
- 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**

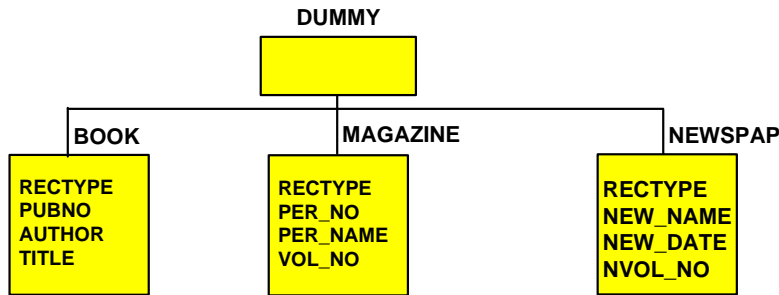
**Describing Unrelated Records Using a Dummy Root Segment**

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

The data source might look like this:



A structure such as the following could 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 our 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 |

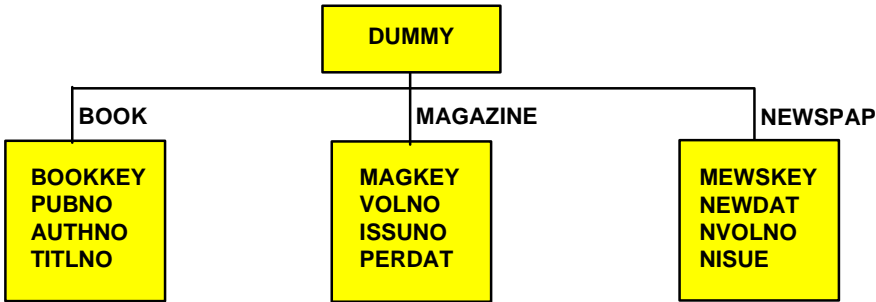
Note the difference between the use of the RECTYPE here and its use when the records are positionally related. In this case, the codes are unrelated and the database designer has chosen to accumulate the records by type first (all the book information together, all the magazine information together, 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 structure:



**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

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 having the common layout. By using a generalized segment—also known as a generalized RECTYPE—instead of one segment per record type, you reduce the number of segments you need to describe in the Master File.

When you use a generalized segment, you identify RECTYPE values using the ACCEPT attribute. You can assign any value you wish 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 VSAM Master Files, 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. In particular, a unique set of field names must be provided for record type M and for record type I, although they have the same layout.

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

```

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

```

## Using an ALIAS in a Report Request

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

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

```
TABLE FILE DOC
PRINT MANUAL MRELEASE MPAGES
BY DOCID BY TITLE BY MDATE
END
PAGE 1
```

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

### Example

#### Using 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

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 | D1 | B1 | B2 | C1 | C1 | D1 | D1 | D1 | D1 | D1 | D1 | D1 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

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

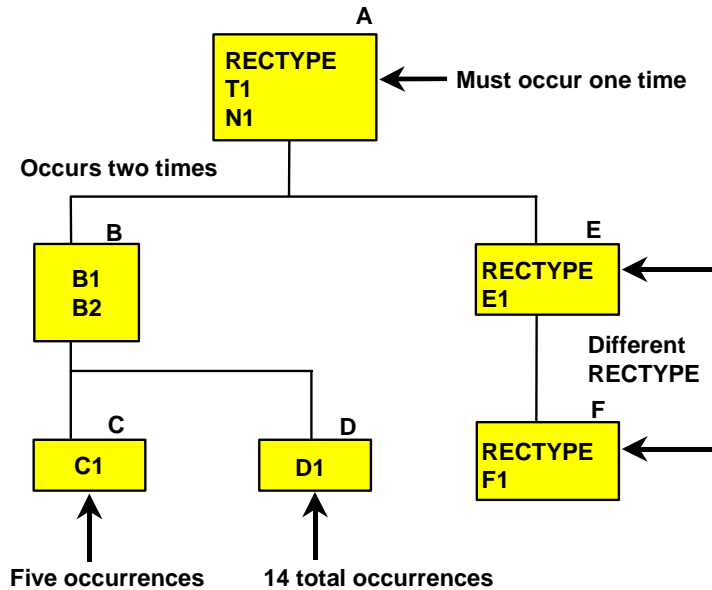
|    |    |
|----|----|
| 03 | F1 |
|----|----|

The data source 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:**

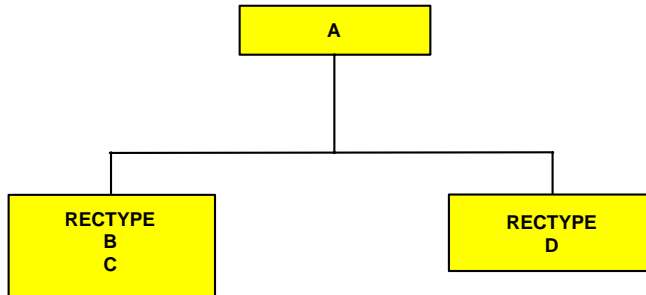
- Segments A, E, and F are different records, related through their record types. The record type attribute consists of certain prescribed values and is stored in a fixed location in the records. The records are expected to be retrieved in a given order. If the first record does not have a RECTYPE of 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).
- 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

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 need to describe three segments in your Master File, with A as the root segment, and segments for B, C, and D as two descendant OCCURS segments for A:



Each of the two descendant OCCURS segments in this example depends on the RECTYPE indicator that appears for each occurrence.

All the rules of syntax for using RECTYPE fields and OCCURS segments also apply to RECTYPES within OCCURS segments.

Since each OCCURS segment depends on the RECTYPE indicator for its evaluation, the RECTYPE must appear at the start of each OCCURS segment. This allows very complex data sources to be described, 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,$
  
```

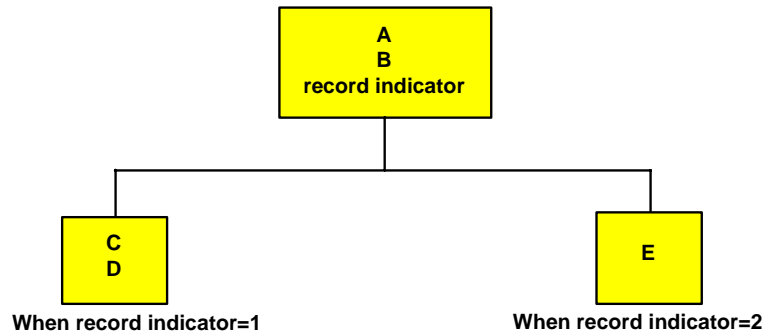
## Describing a Repeating Group Using MAPFIELD

In another possible 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 would appear like this:

|     |                      |     |     |     |
|-----|----------------------|-----|-----|-----|
| A B | record indicator (1) | C D | C D | C D |
| A B | record indicator (2) | E E |     |     |

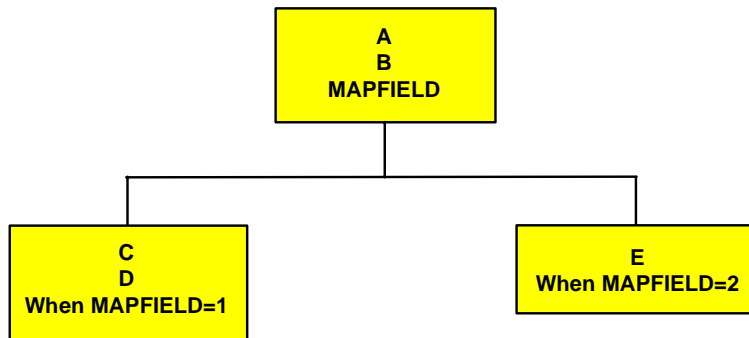
The first record contains header information, values for A and B, followed by an OCCURS segment of C and D that was identified by its preceding record indicator. The second record has a different record indicator and contains a different repeating group, this time for E.

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 RECTYPE does, but since the OCCURS segments will each have their own values for MAPFIELD, the value of MAPFIELD is associated with each OCCURS segment by means of a complementary field named MAPVALUE.

The following diagram illustrates this relationship:



MAPFIELD is assigned as the ALIAS of the field that will be the record indicator. You can give this field any name.

## *Syntax*

### How to Describe a Repeating Group With MAPFIELD

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

where:

*name*

The name you choose to provide for this field.

`ALIAS`

MAPFIELD is assigned as the alias of the field that will be the RECTYPE indicator.

`USAGE`

Follows the usual field format.

`ACTUAL`

Follows the usual field format.

The descendant segment values depend on the value of the MAPFIELD. They are described as separate segments, one for each possible value of MAPFIELD, and all descending from the segment that has the MAPFIELD. A special field, MAPVALUE, is described as the last field in these descendant segments after the ORDER field, if one has been used. The actual MAPFIELD value is supplied as the ALIAS of 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 identifier. If there is an ACCEPT list, this value is any value in the ACCEPT list or range.

**USAGE**

Is the same format as the MAPFIELD format in the parent segment.

**ACTUAL**

Is the same format as the MAPFIELD format in the parent segment.

*list*

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

For example:

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

*range*

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

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

## 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 this 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

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

The AMP sub-parameters BUFND and BUFNI allow MVS BATCH users to enhance the I/O efficiency of TABLE, TABLEF, MODIFY, and JOIN against VSAM 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, job throughput is improved. The new SET commands allow users (in CMS, MVS/TSO, and MSO) to realize similar performance gains in interactive sessions. In general, BUFND (data buffers) increase the efficiency of physical sequential reads, whereas BUFNI (index buffers) are most beneficial in JOIN or KEYED access operations.

### 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. The default values are BUFND=8, BUFNI=1 (eight data buffers and one index buffer).

To determine how many buffers are in effect at any time, issue the query:

```
{MVS|CMS} VSAM SET ?
```

## Using a VSAM Alternate Index

The use of alternate indexes (keys) with VSAM key-sequenced data sources is supported. 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 two components are referred to as the base cluster.

An alternate index is a separate, additional index structure that allows you to access records in a KSDS VSAM 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 have an occasional need to have the records retrieved sorted by job description. The job description field might be described as an alternate index. An alternate index must be related to the base cluster it describes by a path, which is stored in a separate 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 you can take advantage of its potential indirectly, with screening tests (IF...LT, IF...LE, IF...GT, IF...GE, IF...EQ, IF...FROM...TO, IF...IS), which are translated into direct reads against the alternate index. You can also join data sources with the JOIN command through this alternate index.

It is not necessary to explicitly identify the indexed view in order to take advantage of the alternate index. 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 an IF or WHERE test on the alternative index field that meets the above criteria. For example:

```
TABLE FILE CUST
PRINT SSN
WHERE LNAME EQ 'SMITH'
END
```

As you will see in the Master File in *Describing a VSAM Alternate Index* on page 5-43, the LNAME field is defined as an alternate index field. The records in the data source will be retrieved according to their last names, and certain IF screens on the field LNAME will result in direct reads. Note that if the alternate index field name is omitted, the primary key (if there is any) will be used for a sequential or a direct read, and the alternate indexes will be treated as regular fields.



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 alternate indexes are used, but the number of OCCURS segments is unrestricted.

Note that the path name in the allocation will be different from both the cluster name and the alternate index name.

If you are not sure of the path names and alternate indexes associated with a given base cluster, you can use the IDCAMS utility. (See the IBM manual entitled *Using VSAM Commands and Macros* for details.)

### *Example*

### **Describing a VSAM Alternate Index**

Consider the following example:

```
FILENAME = CUST, SUFFIX = VSAM,$
SEGNAME = ROOT, SEGTYPE = S0,$
GROUP = G, ALIAS = KEY, A10, A10,$
  FIELD = SSN, SSN, A10, A10 ,,$
  FIELD = FNAME, DD1, A10, A10, FIELDTYPE=I,$
  FIELD = LNAME, DD2, A10, A10, FIELDTYPE=I,$
```

In this example, SSN is a primary key and FNAME and LNAME are alternate 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 would each have an allocation pointing to the path data set. FNAME and LNAME must have INDEX=I or FIELDTYPE=I coded in the Master File. CUST must be allocated to the base cluster.

## 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

You can read single segment sequential data sources in which fields are separated by any type of delimiter.

## 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
```

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

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

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

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

where:

**DELIMITER**

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

*delimiter*

Identifies a delimiter. For one or more printable characters, the value consists of the actual characters. The delimiter must be enclosed in single quotation marks if it includes characters used as delimiters in Master File syntax. For a non-printable character, the value is the decimal equivalent of the EBCDIC or ASCII representation of the character, depending on your operating environment.

*ufmt*, *afmt*

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

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

## Example

### Defining 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:

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

## 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 FOCUS 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 had 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 will be 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.

---

## CHAPTER 6

# Describing a FOCUS Data Source

### Topics:

- Designing a FOCUS Data Source
- Describing a Single Segment
- Describing an Individual Field
- Describing Two-Gigabyte and Partitioned FOCUS Data Sources

The following topics cover data description topics unique to FOCUS data sources:

- **Design tips.** Provides some suggestions for people who are 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. Chapter 7, *Defining a Join in a Master File*, explains how to define static and dynamic joins in a Master File.
- **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; 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.

# Designing a FOCUS Data Source

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.

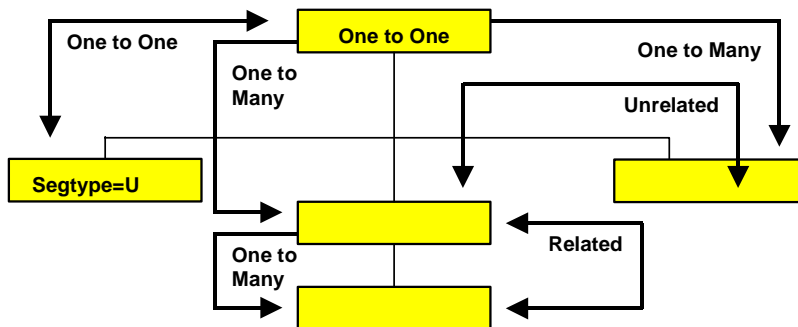
## Data Relationships

The primary consideration when designing a data source is the set of relationships among the various fields. Before you create the Master File, you may wish to draw a diagram of these relationships. Is a field related to any other fields? If so, is it a one-to-one or a one-to-many relationship? If any of the data already exists in another data source, can that data source be joined to this one?

In general, you can use the following guidelines:

- All information that occurs once for a given record should be placed in the root segment or a unique child segment.
- Any information that can be retrieved from a joined data 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—that is, the name of both fields, or the alias of both fields must be identical, or else the name of one field must be identical to the alias of the other.

## General Efficiency Considerations

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 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 need to balance the desire to speed up record *retrieval*, by reducing the size of the root segment, against the need to speed up record selection, by placing fields used in record selection tests as high in the data structure as possible. The segment location of fields used in WHERE or IF tests has significant implications 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 need to 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*. Using alternate views in report requests is discussed in the *Creating Reports* manual.

The following guidelines will help you to design an efficient data structure:

- Limit the information in the root segment to what is necessary to identify the record and to what is used often in screening conditions.
- Avoid unnecessary key fields. Segments with a SEGTYPE of S1 will be processed much more efficiently than those with, for example, a SEGTYPE of S9.
- Index the first field of the segment (the key field) if the root segment of your 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. A SEGTYPE of SH1 puts each new record 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 fields on which you perform frequent searches of unique instances. When you specify that a field be indexed, a table of data values and their corresponding physical locations in the data source is constructed and maintained. 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 will need to weigh the increase in speed against the increase in space.

## Changing a FOCUS Data Source

Once 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 a segment description, you can describe key fields, sort order, and segment relationships.

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*.

## Maximum Number of Segments

The number of segments cannot exceed 64.

## Describing Keys, Sort Order, and Segment Relationships: 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 the highest value to the lowest value, based on the value of the first *n* fields in the segment. *n* can be any number from 1 to 99; if you do not specify it, it defaults to 1.

`S[n]`

Indicates that the segment's instances are sorted from the lowest value to the highest value, based on the value of the first *n* fields in the segment. *n* can be any number from 1 to 255; if you do not specify it, it defaults to 1.

S0

Indicates that the segment has no key 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 will need to be retrieved in the order entered and the application does not need to search for particular instances.

␣ (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 = ␣ 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 ␣ segment.

U

Indicates that the segment is unique—that is, it has a one-to-one relationship to its parent. Note that a unique segment described with a SEGTYPE of U cannot have any children.

KM

Indicates that this is a cross-referenced segment joined to the 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 b or increase the number of key fields. To make any other change to SEGTYPE you need to use the REBUILD facility. REBUILD is described in the *Maintaining Databases* manual.

## Describing a Key Field

You use the SEGTYPE attribute to describe which fields in a segment are key fields. The values of these fields determine how the segment instances are sequenced. The keys must be the first fields in a segment. You can specify up to 255 keys in a segment that is sorted from low to high (SEGTYPE = Sn), and up to 99 keys in a segment sorted from high to low (SEGTYPE = SHn). To maximize efficiency, it is recommended that you specify only as many keys as you need to make each record unique. You can also choose not to have any keys (SEGTYPE = S0 and SEGTYPE = b(blank)).

**Note:** Text fields cannot be used as key fields.

## Describing Sort Order

For segments that have key fields, you use the SEGTYPE attribute to describe the segment's sort order. You can sort a segment's instances in two ways:

- **Low to high.** By specifying a SEGTYPE of  $S_n$  (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 value.
- **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 value.

Segments whose key is a date field often use a high-to-low sort order, since it ensures that the segment instances with the most recent dates will be the first ones encountered in a segment chain.

## 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 will be sorted as follows:

|       |       |       |       |       |
|-------|-------|-------|-------|-------|
| 06345 | 19887 | 19887 | 21334 | 23455 |
| Jones | Smith | Frank | Brown | Walsh |

If you change the field order to put the last name field before the department code and leave SEGTYPE as S1, the last name becomes the key. The segment instances will be sorted as follows:

|       |       |       |       |       |
|-------|-------|-------|-------|-------|
| Brown | Frank | Jones | Smith | Walsh |
| 21334 | 19887 | 06345 | 19887 | 23455 |

Alternately, if you leave the department code as the first field, but set SEGTYPE to S2, the segment will be sorted first by the department code and then by last name, as follows:

|       |       |       |       |       |
|-------|-------|-------|-------|-------|
| 06345 | 19887 | 19887 | 21334 | 23455 |
| Jones | Frank | Smith | Brown | Walsh |

## Describing Segment Relationships

The SEGTYPE attribute describes the relationship of a segment to its parent segment:

- Physical one-to-one relationships are usually specified by setting SEGTYPE to U. If a segment is described in a Master File defined join as descending from the cross-referenced segment, then in the join description, SEGTYPE is set to KLU.
- Physical one-to-many relationships are specified by setting SEGTYPE to any valid value beginning with S (such as S0, 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

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. If you wish, you can use the LOCATION attribute to specify that one or more segments be stored in a physical file separate from the main data source file. You can use a total of 64 LOCATION files per Master File (one LOCATION attribute per segment, except for the root). This can be 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 you may want to use the LOCATION attribute:

- Each physical file is individually subject to a maximum file size of two gigabytes. 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. (You may also want to read Chapter 7, *Defining a Join in a Master File*, to see if it would be 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, the separation of the segments into different data sources allows different disk drives to be used.

Divided data sources do require more careful file maintenance. You have to be especially careful about procedures that are done separately to separate data sources, such as backups. 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.

The LOCATION attribute can be changed if the existing external file name (ddname) can be changed.

**Syntax**

**How to Store a Segment in a Different Location**

`LOCATION = filename`

where:

`filename`

Is the ddname of the file in which the segment is to be stored.

For example:

`SEGNAME = HISTREC, SEGTYPE = S1, PARENT = SSNREC, LOCATION = HISTFILE, $`

**Example**

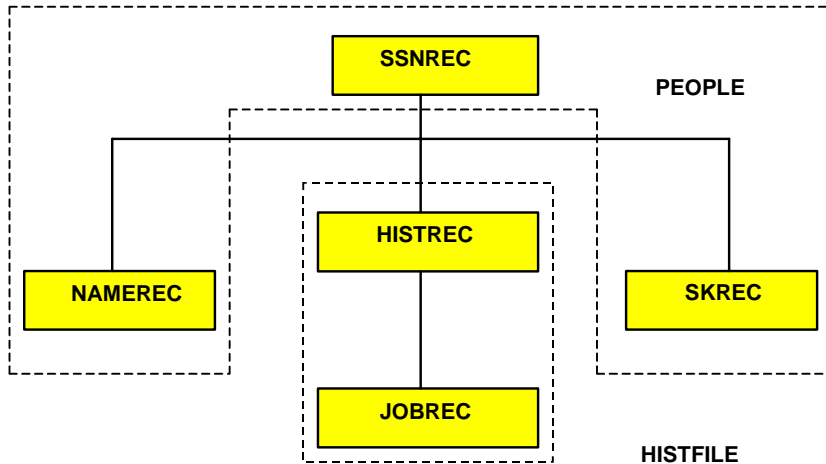
**Specifying Location for a Segment**

The following example 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. This occurs because if no LOCATION attribute is specified for a segment, it is placed by default in the same file as its parent. In our example, you could assign the SKREC segment to a different file by specifying the LOCATION attribute in its declaration. However, it is strongly recommended that the LOCATION attribute be specified and not allowed 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. You 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 will be 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. That is, you can use the LOCATION attribute for a text field without using it for a segment. You can also use the LOCATION attribute for both the segment and the text field in the same Master File.

**Note:** Field names for text fields in FOCUS Master Files are limited to 12 characters; however, alias names for these fields can be up to 66 characters.

## Limits on the Number of LOCATION Files

There is a limit on the number of different location segments and text location files you can specify. This limit is based on the number of entries allowed in the file directory table (FDT) for FOCUS 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. The FDT can contain 189 entries of which up to 64 can represent segments and location files. Each unique location file counts as one entry in the FDT.

For a given FOCUS data source, the maximum number of location files can be determined by the following formula

*Available FDT entries* = 189 - (*Segments* + *Indexes*)

*Location files* = min (64, Available FDT entries)

where:

*Location files*

Is the maximum number of location segments and text location files (up to a maximum of 64).

*Segments*

Is the number of segments in the Master File.

*Indexes*

Is the number of indexed fields.

For example, a ten-segment data source with 2 indexed fields would enable you to specify up to 52 location segments and/or location files for text fields (189 - (10 + 2)). Using the formula, the result would equal 177; however, the maximum number of text location files must always be *no more than 64*.

**Note:** If you specify a text field with a LOCATION attribute, the main file will be included in the text location file count.

## Timestamping a FOCUS Segment: 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 will not timestamp the field. It will 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 will be 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 that 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 will be 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 = name, ALIAS = alias, {USAGE|FORMAT} = AUTODATE , $
```

where:

*name*

Is any valid field name.

*alias*

Is any valid alias.

## Example

### Defining an AUTODATE Field

The EMPDATE data source can be created 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
```

You can 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/HYMD = 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* will 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 \* will 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/time per transaction.
- SU updates the AUTODATE field per segment using the date and time on the FOCUS Database Server.
- Maintain will process AUTODATE fields at COMMIT time.
- DBA is permitted on the AUTODATE field; however, when unrestricted fields in the segment are updated, the system will update the AUTODATE field.
- The AUTODATE field does not support the following attributes: MISSING, ACCEPT, and HELPMESSAGE.

## Describing an Individual Field

There are two field attributes that have special values or are unique to FOCUS data sources: ACCEPT and INDEX (also known as FIELDTYPE). This section describes both of these, and also documents the internal formats of each FORMAT data type and the internal values with the MISSING attribute.

### The ACCEPT Attribute

ACCEPT is an optional attribute that you can use to validate data that is entered into a field using a MODIFY procedure. Its use with all types of data sources is described in 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. This is described in Chapter 4, *Describing an Individual Field*.

*range*

Gives the range of acceptable values. This is described in 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.

AS *targetfield*

Is the name of the field that contains the acceptable data values. This field must be indexed.

IN *file*

Is the name of the Master File describing the data source that contains the indexed field of acceptable values.

## The INDEX Attribute

You can 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 if you want to:

- Join two segments. The cross-referenced field in a joined FOCUS data source must be indexed, as described in *Describing Single Segments* on page XX (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
```

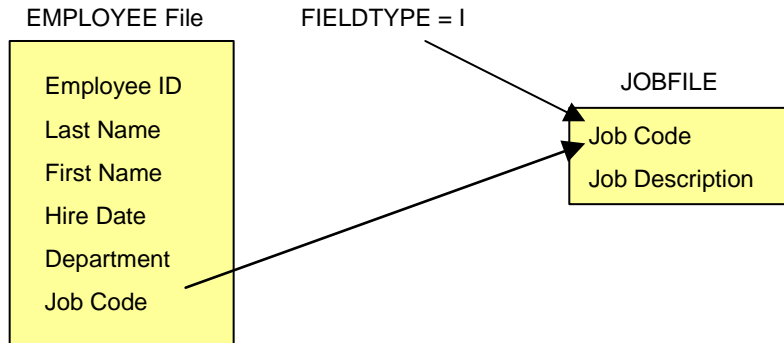
Text fields cannot be indexed. The maximum field name length for indexed fields is 12 characters.

For example:

```
FIELDNAME = JOBCODE, ALIAS = CJC, FORMAT = A3, INDEX = I, $
```

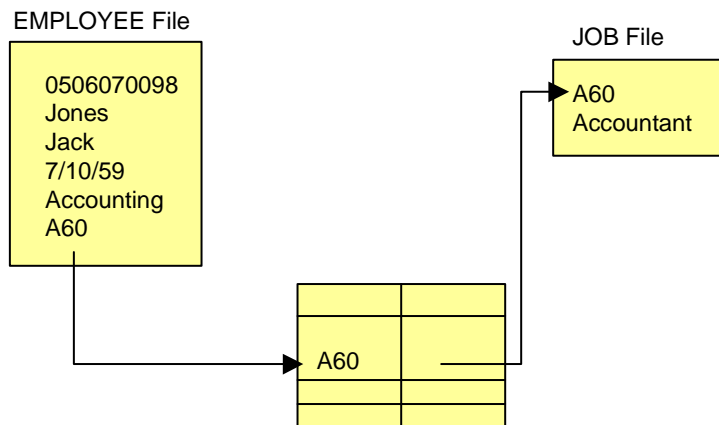
## Joins and the INDEX Attribute

In order for a segment to be cross-referenced with a static cross-reference, a dynamic cross-reference, or a JOIN, at least one field in the cross-referenced 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 a value of blank, which is equivalent, the index will no longer be maintained. If you no longer need the index, after you remove the INDEX attribute use the REORG option of the REBUILD facility to recover space occupied by the index. REBUILD is described in the *Maintaining Databases* manual.

If you wish to turn off indexing temporarily—for example, to load a large amount of data into the data source quickly—you can remove the INDEX attribute before loading the data, restore the index 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. The following diagnostic message is issued if you attempt this:

```
(FOC720) THE NUMBER OF INDEXES ADDED AFTER FILE CREATION EXCEEDS 7
```

- **Maximum number.** The combined 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).
- Alphanumeric fields are stored as characters in the specified number of bytes.
- Missing values are represented internally by a dot (.) for alphanumeric fields, and as the value -9998998 for numeric fields.



# Describing Two-Gigabyte and Partitioned FOCUS Data Sources

The FOCUS data source size can be a maximum of two gigabytes per physical data file. Through partitioning, one logical FOCUS data source can now span up to 500 gigabytes. 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.

## Partitioning a FOCUS Data Source

FOCUS data sources 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. The horizontal partition is a slice of the entire data source structure. Note, however, that the number of physical files associated with one FOCUS data source is the sum of all of its partitions and LOCATION files. This sum must be less than or equal to 250. FOCUS data sources can grow in size over time, and can be repartitioned based on the requirements of the application.

**Note:** You do not have to partition your data source. If you choose not to, your application will automatically support 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 horizontal partition contains the complete data source structure for specific data values or ranges of values. Intelligent partitioning lets you not only separate the data into up to 250 physical two-gigabyte files, it allows you to create an Access File in which you describe, using WHERE criteria, the actual data values in each partition. When processing a report request, the selection criteria in the request are compared to the WHERE criteria in the Access File to determine which partitions are required for retrieval.

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

To take advantage of the partitioning feature, you must:

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

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

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

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

## Specifying an Access File in a FOCUS Master File

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

### *Syntax*      How to Specify an Access File for a Partitioned FOCUS Data Source

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

where:

*fname*

Is the file name of the partitioned data source.

*accessfile*

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

### *Example*      Master File for the VIDEOTR2 Partitioned 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=HYMDI, $
SEGNAME=SALES, SEGTYPE=S2, PARENT=TRANSDAT
FIELDNAME=TRANSCODE, ALIAS=TCOD, FORMAT=I3, $
FIELDNAME=QUANTITY, ALIAS=NO, FORMAT=I3S, $
FIELDNAME=TRANSTOT, ALIAS=TTOT, FORMAT=F7.2S, $
SEGNAME=RENTALS, SEGTYPE=S2, PARENT=TRANSDAT
FIELDNAME=MOVIECODE, ALIAS=MCOD, FORMAT=A6, INDEX=I, $
FIELDNAME=COPY, ALIAS=COPY, FORMAT=I2, $
FIELDNAME=RETURNDATE, ALIAS=INDATE, FORMAT=YMD, $
FIELDNAME=FEE, ALIAS=FEE, FORMAT=F5.2S, $
DEFINE DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
```

## The FOCUS Access File

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

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

## FOCUS Access File Attributes

The Access File can include the following attributes:

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

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

**Syntax****How to Create an Access File**

```

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

```

where:

**MASTERNAME**

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

*filename1, filename2*

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

**DATANAME**

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

*dataname1, dataname2*

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

*test*

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

*fieldname relational\_operator value1 [OR value2 OR value3 ... ]*

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

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

where:

*fieldname, fieldname1, fieldname2*

Are field names in the Master File.

*relational\_operator*

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

*value1, value2, value3, value4*

Are valid values for their corresponding fields.

**Note:** If the test conditions do not accurately reflect the contents of the 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 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 MVS:

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

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

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

For CMS:

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

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

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

### Example

### Describing an Intelligent Partition With a LOCATION 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 MVS:

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

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

For CMS:

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

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

### **Example**      **Using a Partitioned Data Source**

The following illustrates how to use a partitioned data source:

```
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 was used. However, only the second partition is retrieved, reducing I/O and enhancing performance.

## **Describing Joined 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

Recall that 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 MVS:

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

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

For CMS:

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

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

## Reference

### Usage Notes for a Two-Gigabyte FOCUS Data Source

- 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.
- To sort a FOCUS data source that is larger than one gigabyte, on MVS you must explicitly allocate ddname FOCSORT to a temporary file with enough space to contain the data; on VM, you must have enough TEMP space available.
- To REBUILD a FOCUS data source that is larger than one gigabyte, on MVS you must explicitly allocate ddname REBUILD to a temporary file with enough space to contain the data; on VM you must have enough TEMP space available. It is strongly recommended that you REBUILD/REORG in sections, to a new file, 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 MVS, you must issue the LOAD command with the following syntax:

`LOAD NOCREATE`

- If you create a FOCUS data source that is larger than one gigabyte using HOLD FORMAT FOCUS, on MVS you must explicitly allocate ddnames FOC\$HOLD and FOCSORT to temporary files large enough to hold the data; on VM you must have enough TEMP space available.
- The order of precedence for allocating data sources is as follows:
  - A USE command in effect has the highest precedence. It overrides an Access File or an explicit allocation for a data source.
  - An Access File overrides an explicit allocation for a data source.
- A DATASET attribute cannot be used in the same Master File as an ACCESSFILE attribute.

---

## CHAPTER 7

# Defining a Join in a Master File

### 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 Master File Defined Joins and the JOIN Command
- Joining to One Cross-Referenced Segment From Several Host Segments

You can describe a new relationship between any two segments that have at least one field in common by joining them. The underlying data structures remain physically separate, but FOCUS treats them as if they were part of a single structure from which you can report. This chapter describes how to define a join in a Master File for FOCUS, 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.

## 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 FOCUS session (or until you clear it during the session) and creates a temporary view of the data that includes all of the segments in both 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 7-3. This is supported for FOCUS data sources only.

### Development Tip:

Some users find it helpful to prototype a database design first using dynamic joins—implemented by issuing the JOIN command or within the Master File—and, once the design is stable, to change the frequently-used joins to static joins defined in the Master File, accelerating 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, as described in the *Maintaining Databases* manual.

**Note:** Master File defined joins are sometimes referred to as cross-references.

## Static Joins Defined in the Master File: SEGTYPE = KU and KM

Static joins allow you to relate segments in different FOCUS data sources permanently. You specify static joins in the Master File of the host data source.

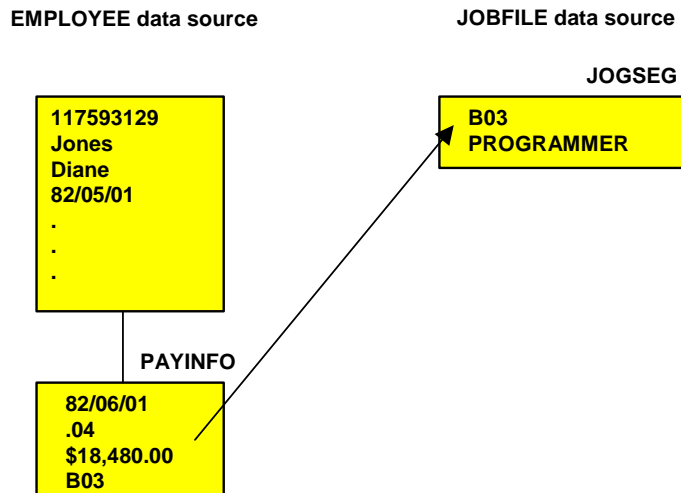
There are two types of static joins: one-to-one (SEGTYPE KU) and one-to-many (SEGTYPE KM).

- 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 JOBFIL. You can retrieve the job description from JOBFIL by locating the record whose JOBCODE corresponds to the JOBCODE value in the EMPLOYEE data source, as shown in the following diagram:

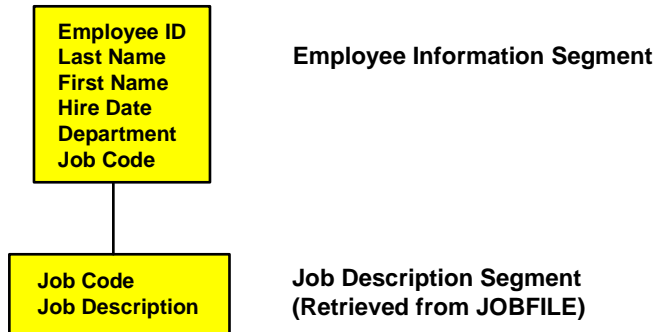


Using a join in this situation saves you the trouble of entering and revising the job description for every record in the EMPLOYEE data source. Instead, you can maintain a single list of valid job descriptions in the JOBFIL data source. Changes need be made only once, in JOBFIL, and are reflected in 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 FOCUS treats the EMPLOYEE data source as though it also contains the Job Description segment from the JOBFIL data source. The actual structure of the JOBFIL data source is not affected. FOCUS will view the EMPLOYEE data source as follows:

**EMPLOYEE File**



## Syntax

### How to Specify a Static Unique Join

```
SEGNAME = segname, SEGTYPE = KU, PARENT = parent,  
CRFILE = db_name, CRKEY = field, [CRSEGNAME = crsegname,] $
```

where:

*segname*

Is the name by which the cross-referenced segment will be known in the host 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 and/or alias) of the host field and the cross-referenced field. The field name or alias of the host field must be identical to the field name of the cross-referenced field. You can change the field name without rebuilding the 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.

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 (for simplicity, fields and segments not essential to the example are not shown):

```
FILENAME = EMPLOYEE, SUFFIX = FOC, $  
  
SEGNAME = EMPINFO, SEGTYPE = S1, $  
.  
.  
.  
SEGNAME = PAYINFO, SEGTYPE = SH1, PARENT = EMPINFO, $  
      FIELDNAME = JOBCODE, ALIAS = JBC, FORMAT = A3, $  
.  
.  
.  
SEGNAME = JOBSEG, SEGTYPE = KU, PARENT = PAYINFO, CRFILE = JOBFILE,  
      CRKEY = JOBCODE, $
```

Note that you only have to give the name of the cross-referenced segment; the fields in that segment are already known from the cross-referenced 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**

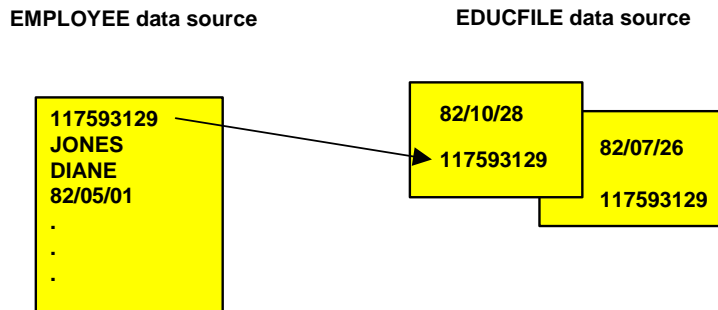
Decoding is the process of matching a code (such as the job code in our example) to the information it represents (such as the job description). Because every code has only one set of information associated with it, the join between the code and the information should be one-to-one, that is, unique. You can decode using a join, as in our example, or using the DECODE function with the DEFINE FILE command, as described in the *Creating Reports* manual. The join method is recommended when there are a large number of codes.



## Describing a Non-Unique Join: SEGTYPE = KM

You use a one-to-many join (that is, a non-unique join) when you have several instances of data in the cross-referenced segment associated with a single instance in the host segment. Using our EMPLOYEE example, suppose that you kept an educational data source named EDUCFILE to track the course work employees were doing. 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 7-5, except that you supply a different value, KM (which stands for keyed multiple), for the SEGTYPE attribute, as follows:

```
SEGTYPE = KM
```

**Example**

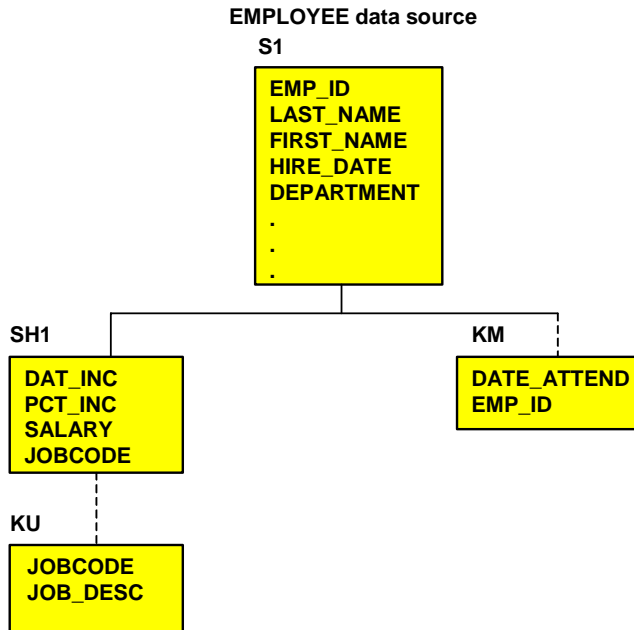
**Specifying a Static Multiple Join**

```
SEGNAME = ATNDSEG, 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 = ATNDSEG, SEGTYPE = KM, PARENT = EMPINFO, CRFILE = EDUCFILE,  
        CRKEY = EMP_ID, $
```

Within a report request, FOCUS treats both cross-referenced data sources, JOBFILE and EDUCFILE, 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

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,] $
```

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.

**Example**

### Identifying Cross-Referenced Descendant Segments

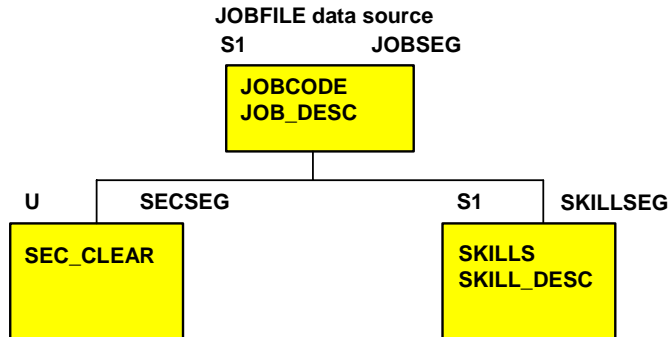
```
SEGNAME = SECSEG, SEGTYPE = KLU, PARENT = JOBSEG, CRFILE = JOBFILE, $  
SEGNAME = SKILLSEG, SEGTYPE = KL, PARENT = JOBSEG, CRFILE = JOBFILE, $
```

Note that you do not use the CRKEY attribute in a declaration for a linked segment, since the common join field (which is identified by CRKEY) only needs to be specified for the cross-referenced segment.

**Example**

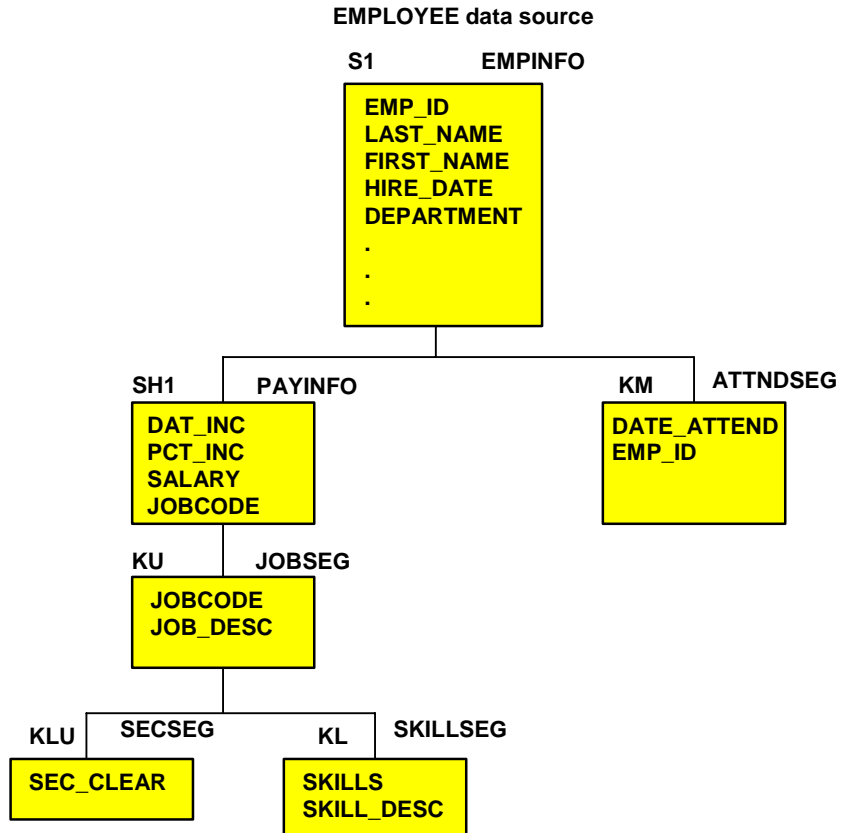
### Using Cross-Referenced Descendant Segments

Consider our EMPLOYEE example. JOBFILE is a multi-segment 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. Once 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 FOCUS retrieves the JOBSEG segment from JOBFILE, it also retrieves all of JOBSEG's children that were declared with KL or KLU SEGTYPEs in the EMPLOYEE Master File:



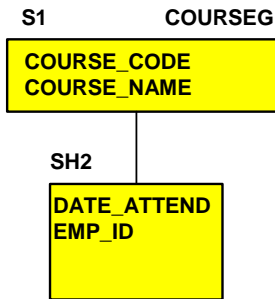
**Example**

**Using Cross-Referenced Ancestral Segments**

Remember that you can retrieve all of the segments in a cross-referenced 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.

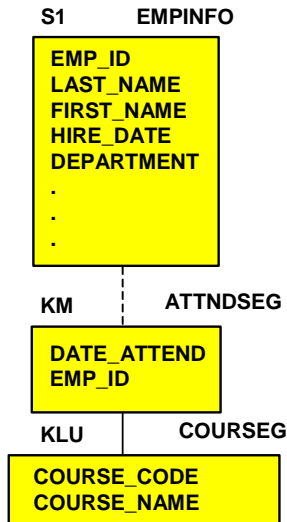
Consider the EDUCFILE data source used in our example. The COURSESEG segment is the root and describes each course; ATTNDSEG is a child and includes employee attendance information:

**EDUCFILE data source**



When you join EMPINFO in EMPLOYEE to ATTNDSEG in EDUCFILE, you can access course descriptions in COURSESEG by declaring it as a linked segment. From this perspective, COURSESEG is a child of ATTNDSEG:

**EMPLOYEE data source**

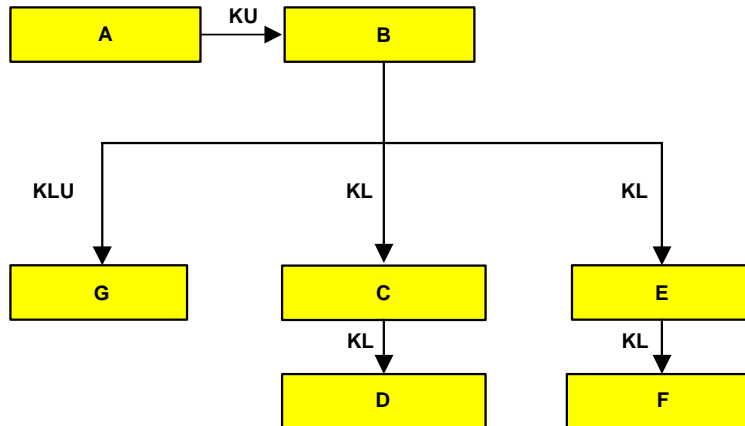


The sections of the EMPLOYEE Master File used in our examples follow (nonessential fields and segments are not shown):

```
FILENAME = EMPLOYEE, SUFFIX = FOC, $  
  
SEGNAME = EMPINFO, SEGTYPE = S1, $  
    FIELDNAME = EMP_ID, ALIAS = EID, FORMAT = A9, $  
    .  
    .  
    .  
SEGNAME = PAYINFO, SEGTYPE = SH1, PARENT = EMPINFO, $  
    FIELDNAME = JOBCODE, ALIAS = JBC, FORMAT = A3, $  
    .  
    .  
    .  
SEGNAME = JOBSEG,   SEGTYPE = KU,   PARENT = PAYINFO,  CRFILE = JOBFILE,  
    CRKEY = JOBCODE, $  
  
SEGNAME = SECSEG,   SEGTYPE = KLU,  PARENT = JOBSEG,   CRFILE = JOBFILE, $  
SEGNAME = SKILLSEG, SEGTYPE = KL,   PARENT = JOBSEG,   CRFILE = JOBFILE, $  
SEGNAME = ATTNDSEG, SEGTYPE = KM,   PARENT = EMPINFO,  CRFILE = EDUCFILE,  
    CRKEY = EMP_ID, $  
SEGNAME = COURSEG,  SEGTYPE = KLU,  PARENT = ATTNDSEG, CRFILE = EDUCFILE, $
```

## Hierarchies of Linked Segments

A KL segment may lead to other KL segments. Graphically, this can be illustrated as:



The letters on the arrows are the SEGTYPEs.

Note that segment G may either be a unique descendant of B or B's parent.



## Dynamic Joins Defined in the Master File: SEGTYPE = DKU and DKM

You can define a dynamic join in a Master File using the SEGTYPE attribute. There are two types of dynamic Master File defined joins: one-to-one (SEGTYPE DKU) and one-to-many (SEGTYPE DKM).

- As with a static join, you specify a one-to-one join, also known as a unique join, when you want to retrieve at most one record instance from the cross-referenced 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 has to do with storage, speed, and flexibility:

- The links (pointers) for a static join are retrieved once and then permanently stored in the host 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 only redefine or remove a static join using the REBUILD facility, as described in the *Maintaining Databases* manual. You can redefine or remove a dynamic join at any time by editing the Master File.

### *Syntax*

### How to Specify a Dynamic Join 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 7-5), 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 = JOBFIL, 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 Master File Defined Joins and the JOIN Command

If you wish 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 they are needed. This makes static joins faster than dynamic ones, since the links only need to be established once, but less flexible, as you can only redefine or remove a static join by using the REBUILD facility. The REBUILD facility is described in the *Maintaining Data* manual.

Among dynamic joins the JOIN command is easier to use in that you do not need to edit the Master File each time you want to change the join specification, and you do not need to describe each linked segment as it appears from the perspective of the host 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.

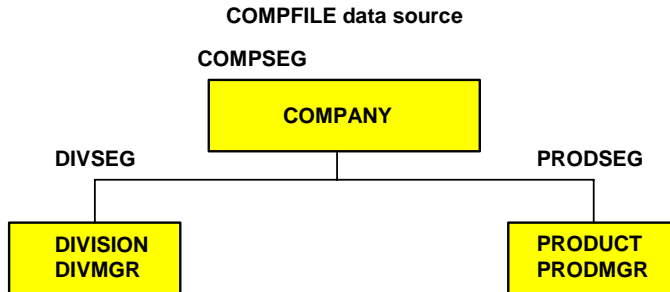
|                                                              | <b>Advantages</b>                                                                                                                                                                                                                         | <b>Disadvantages</b>                                                                                                                                                                                                                                               |
|--------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Static Join in Master File</b><br>(SEGTYPE = KU or KM)    | Faster after first use: links are created only once.<br><br>Always in effect.<br><br>Can select some linked segments and omit others.                                                                                                     | Must be specified before data source is created or reloaded using REBUILD.<br><br>Requires REBUILD utility to change.<br><br>Requires four bytes of file space per instance.<br><br>User needs to know how to specify relationships for linked segments (KL, KLU). |
| <b>Dynamic Join in Master File</b><br>(SEGTYPE = DKU or DKM) | Can be specified at any time.<br><br>Always in effect. Does not use any space in the data source.<br><br>Can be changed or removed as needed, without using the REBUILD facility.<br><br>Can select some linked segments and omit others. | Slower: links are retrieved for each record in each report request.<br><br><br><br>User needs to know how to specify relationships for linked segments (KL, KLU).                                                                                                  |
| <b>Dynamic Join (using the JOIN Command)</b>                 | Can be specified at any time.<br><br>Does not use any space in the data source. Can be changed or removed as needed, without using the REBUILD facility.<br><br>User never needs to describe relationships of linked segments.            | Slower: links are retrieved for each record in each report request.<br><br>JOIN command must be issued in each session in which you want the join to be in effect.<br><br>All linked segments are always included, whether or not you need them.                   |

## Joining to One Cross-Referenced Segment From Several Host Segments

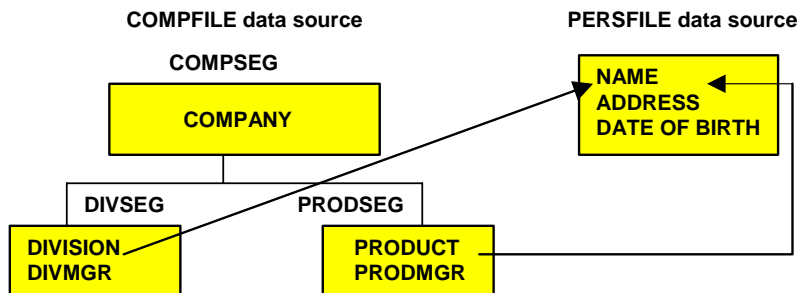
You may come upon situations where you need to join to one cross-referenced segment from several different segments in the host data source. You may also find a need to join to one cross-referenced segment from two different host data sources at the same time. You can handle these data structures using 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. You might want to 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.

FOCUS provides a way 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 might look like this:

```
FILENAME = PERSFILE, SUFFIX = FOC, $
SEGNAME = IDSEG, SEGTYPE = S1, $
    FIELD = NAME, ALIAS = FNAME, FORMAT = A12, INDEX=I, $
    FIELD = ADDRESS, ALIAS = DAS, FORMAT = A24, $
    FIELD = DOB, ALIAS = IDOB, FORMAT = YMD, $
```

You use the following Master File to join PERSFILE to COMPFIL. Note that there is no record terminator (\$) following the cross-referenced segment declaration (preceding the cross-referenced field declarations).

```
FILENAME = COMPFIL, SUFFIX = FOC, $
SEGNAME = COMPSEG, SEGTYPE = S1, $
    FIELD = COMPANY, ALIAS = CPY, FORMAT = A40, $
SEGNAME = DIVSEG, PARENT = COMPSEG, SEGTYPE = S1, $
    FIELD = DIVISION, ALIAS = DV, FORMAT = A20, $
    FIELD = DIVMGR, ALIAS = NAME, FORMAT = A12, $
SEGNAME = ADSEG, PARENT = DIVSEG, SEGTYPE = KU,
    CRSEGNAME = IDSEG, CRKEY = DIVMGR, CRFILE = PERSFILE,
    FIELD = NAME, ALIAS = FNAME, FORMAT = A12, INDEX = I, $
    FIELD = DADDRESS, ALIAS = ADDRESS, FORMAT = A24, $
    FIELD = DDOB, ALIAS = DOB, FORMAT = YMD, $
SEGNAME = PRODSEG, PARENT = COMPSEG, SEGTYPE = S1, $
    FIELD = PRODUCT, ALIAS = PDT, FORMAT = A8, $
    FIELD = PRODMGR, ALIAS = NAME, FORMAT = A12, $
SEGNAME = BDSEG, PARENT = PRODSEG, SEGTYPE = KU,
    CRSEGNAME = IDSEG, CRKEY = PRODMGR, CRFILE = PERSFILE,
    FIELD = NAME, ALIAS = FNAME, FORMAT = A12, INDEX = I, $
    FIELD = PADDRESS, ALIAS = ADDRESS, FORMAT = A24, $
    FIELD = PDOB, ALIAS = DOB, FORMAT = YMD, $
```

DIVMGR and PRODMGR are described as CRKEYs. FOCUS automatically matches their common alias, NAME, to the field name NAME in the PERSFILE data source. In addition, the field declarations that follow the join information rename the ADDRESS and DOB fields so that they can be referred to separately in reports. Their actual field names in the PERSFILE are supplied as aliases.

Note that the NAME field cannot be renamed, since it is the common join field. It must be included in the declaration along with the fields being renamed, as it is described in the cross-referenced data source. That it cannot be renamed is not a problem, since its ALIAS can be renamed, and, in any event, the field does not need to be used in reports: because it is the join field, it contains exactly the same information as the DIVMGR and PRODMGR fields.

The following conventions must be observed:

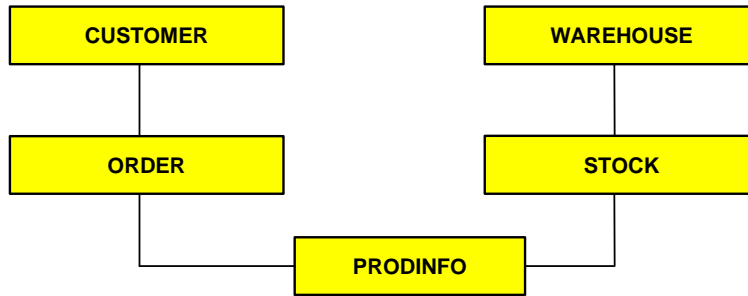
- The common join field's FIELDNAME or ALIAS in the host data source must be identical to its FIELDNAME in the cross-referenced data source.
- The common join field should not be renamed, but the alias can be changed. The other fields in the cross-referenced segment can be renamed.
- Place field declarations 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

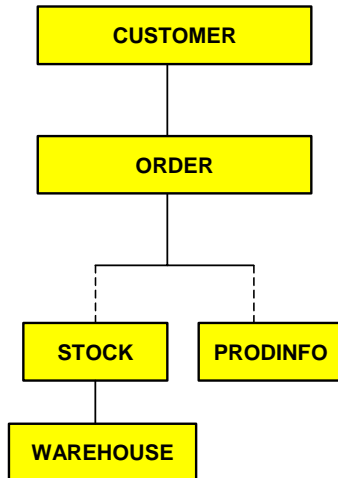
At some point you may need to join to a cross-referenced segment from two different host data sources at the same time. If you were to describe a structure like this as a single data source, you would have to have two parents for the same segment, which is invalid. You can, however, describe the information 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:

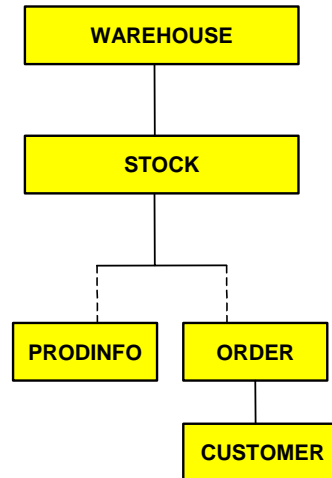


You can join several data sources to create this structure. For example:

view from ORDERS data source:



view from INVENTORY data source:





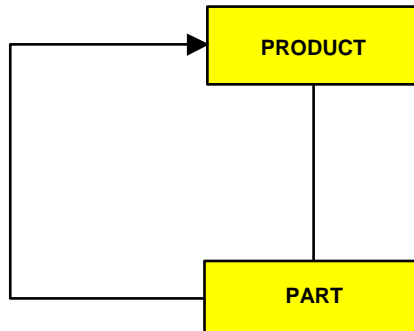
The CUSTOMER and ORDER segments are in the ORDERS data source, the WAREHOUSE and STOCK segments are in the INVENTORY data source, and the PRODINFO segment is stored in the PRODUCTS data source. Both the INVENTORY and ORDERS data sources have one-to-one joins to the PRODUCTS data source. In the INVENTORY 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 INVENTORY 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 INVENTORY data source.

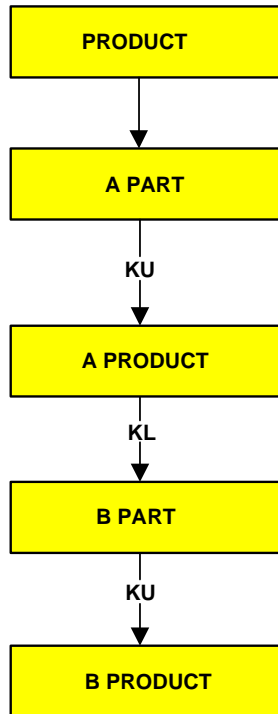
The joins among these three data sources can be viewed from the perspectives of both host data sources, approximating the multiple-parent structure described earlier.

## 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 sub-parts. Schematically, this would appear as:



A FOCUS description for this case, shown for two levels of sub-parts, is:



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

---

## CHAPTER 8

# Checking and Changing a Master File: CHECK

### Topics:

- CHECK Command Display
- The PICTURE Option
- The HOLD Option

Use the CHECK command to validate your Master Files. You must always do this after writing the Master File. If you do not issue the CHECK command, FOCUS may not update your Master File with the changes that you just made. The CHECK output highlights any errors in your Master File and allows you to correct them before reading the data source. After making any necessary corrections, use CHECK again to confirm that the Master File is valid.

## Syntax

## How to Check Data Source Descriptions

```
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 *The PICTURE Option*, on page 8-5.

RETRIEVE

Alters the picture to reflect the order in which segments are retrieved when TABLE or TABLEF commands are issued. Note that unique segments are viewed as logical extensions of their parent segment. The keyword RETRIEVE can be abbreviated to RETR.

DUPLICATE

Lists duplicate field names for the specified 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 *The HOLD Option*, on page 8-8.

*name*

Is a name for the HOLD file and HOLD Master File.

ALL

Adds the values of FDEFCENT and FYRTHRESH at the file level and the values of DEFCENT and YRTHRESH at the field level to the HOLD file.

# CHECK Command Display

If your Master File contains syntactical errors, the CHECK command displays appropriate error messages.

If the 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 displays 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

For example, entering the following command

```
CHECK FILE EMPLOYEE
```

produces the following information:

```
NUMBER OF ERRORS=          0
NUMBER OF SEGMENTS=       11    ( REAL=          6 VIRTUAL=          5 )
NUMBER OF FIELDS=         34    INDEXES=          0 FILES=          3
TOTAL LENGTH OF ALL FIELDS = 365
```

When you are using FOCUS online, this message 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 that displays near the top of your screen to verify the accuracy of the field lengths you have 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 by FOCUS.

When the CHECK command is issued for a data source that has more than one field of the same name within the same segment, a FOC1829 error message is generated along with a warning indicating the duplicate field names, such as the following:

```
(FOC1829) WARNING. FIELDNAME IS NOT UNIQUE WITHIN A SEGMENT: BB
WARNING: FOLLOWING FIELDS CANNOT BE ACCESSED
BB IN SEGMENT SEGA          (VS SEGB )
```

When the DUPLICATE option is added, the output contains a warning message like the following:

```
WARNING: FOLLOWING FIELDS APPEAR MORE THAN ONCE
AA IN SEGMENT SEGB          (VS SEGA )
```

# The PICTURE Option

The PICTURE option displays a diagram of the FOCUS structure defined by the Master File. Each segment is represented by a box. There are four types of boxes, which indicate whether a segment (including the root segment) is non-unique or unique and whether it is real or cross-referenced. The four types of boxes are:

## Real segments

### Non-unique segment:

```

      segname
num      segtype
*****
*field1  **I
*field2  **
*field3  **
*field4  **
*        **
*****
*****

```

### Unique segment:

```

      segname
num      U
*****
*field1  *I
*field2  *
*field3  *
*field4  *
*        *
*****

```

## Cross-referenced segments

### Non-unique segment:

```

      segname
num      KM (or KLM)
:.....:
:field1  ::K
:field2  ::
:field3  ::
:field4  ::
:        ::
:.....:
:.....:
      crfile

```

### Unique segment

```

      segname
num      KU (or KLU)
:.....:
:field1  :K
:field2  :
:field3  :
:field4  :
:        :
:.....:
      crfile

```

where:

*num*

Is the number assigned to the segment in the FOCUS structure.

*segname*

Is the name of the segment.

*segtype*

Is the segment type for a real, non-unique segment: Sn, SHn, or N (for blank segtypes).

*field1 ...*

Are the names of fields in the segment. Field names of 66 characters are truncated to 12 characters in CHECK FILE PICTURE operations.

I

Indicates an indexed field.

K

Indicates the key field in the cross-referenced segment.

*crfile*

Is the name of the cross-referenced 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 02/08/99 AT 12.33.04

          JOBSEG
01          S1
*****
*EMP_ID      **
*FIRST_NAME  **
*LAST_NAME   **
*JOB_TITLE   **
*            **
*****
          I
          I
          I
          I SALSEG
02          I KU
.....
:EMP_ID      :K
:SALARY      :
:EXEMPTIONS  :
:            :
:            :
:.....:
JOINED SALARY

```

## The HOLD Option

The HOLD option generates a temporary HOLD file. HOLD files are explained in the *Creating Reports* manual. This HOLD file contains detailed information regarding file, segment, and field attributes, which you can display in reports using TABLE requests.

Certain fields in this HOLD file are of special interest. Unless otherwise noted, these fields are named the same as attributes in Master Files; each field stores the values of the similarly-named attribute. The fields can be grouped into file attributes, segment attributes, and field attributes.

### File Attributes:

FILENAME

SUFFIX

FDEFCENT, FYRTHRESH

Note that these attributes are included in the HOLD file, if they exist in the original Master File and you specify the ALL option.

### Segment Attributes:

SEGNAME

SEGTYPE

Note that this field does not indicate the number of segment key fields. Segment types S1, S2, and so on are shown as type S. The same is true with segment type SHn.

SKEYS

The number of segment key fields. For example, if the segment type is S2, SKEYS has the value 2.

SEGNO

The number assigned to the segment within the FOCUS structure. This is displayed in the picture.

LEVEL

The level of the segment within the FOCUS structure. The root segment is on Level 1, its children are on Level 2, and so on.

PARENT

CRKEY

FIELDNAME

**Field Attributes:**

ALIAS

FORMAT

ACTUAL

Note that if you include the FORMAT field in the TABLE request, you should not use the full field name FORMAT. Rather, you should use the alias USAGE or a unique truncation of the FORMAT field name (the shortest unique truncation is FO).

DEFCENT, YRTHRESH

Note that these attributes are included in the HOLD file, if they exist in the original Master File and you specify the ALL option.

**Example****Using the CHECK FILE HOLD Option**

This sample FOCUS procedure creates a HOLD file describing the EMPLOYEE data source. It then writes a report that displays the names of cross-referenced segments in the EMPLOYEE data source, their segment types, and the attributes of their fields: field names, aliases, and formats.

```
CHECK FILE EMPLOYEE HOLD
TABLE FILE HOLD
HEADING
"FIELDNAMES, ALIASES, AND FORMATS"
"OF CROSS-REFERENCED FIELDS IN THE EMPLOYEE DATA SOURCE"
" "
PRINT FIELDNAME ALIAS 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 AS Names With the HOLD Option

An AS name may be provided for the temporary HOLD file generated by the CHECK command. If a name is not specified, the default name is HOLD and FOCUS will replace any existing default file.

### Note:

- The AS name may not be longer than 8 characters, or FOCUS defaults to the name HOLD and no warning is issued.
- When the AS option is specified in combination with other CHECK options, the AS holdname specification must appear last.

## TITLE, HELPMESSAGE, and TAG Attributes

When you use the HOLD option of the CHECK command, FOCUS places the TITLE text in the TITLE field of the FLDATTR segment, the HELPMESSAGE text in the HELPMESSAGE field of the FLDATTR segment, and the TAG names in the TAGNAME field of the SEGATTR segment.

When no JOINS are in effect, or when a JOIN command is issued without a TAG name, the TAGNAME field by default contains the name of the 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 be displayed as blank in reports. FLDSEG may be dynamically reformatted in a TABLE request (FLDSEG/I2) to force the display of zero.

Once data has been entered into a FOCUS 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.

---

## CHAPTER 9

# Accessing a FOCUS Data Source: USE

### Topics:

- The USE Command
- Specifying a Non-Default File ID
- Identifying New Data Sources to FOCUS
- Accessing Data Sources in Read Only Mode
- Concatenating Data Sources
- Accessing Simultaneous Usage Data Sources
- Using the LOCATION Attribute
- Displaying the USE Options in Effect

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*.

## The USE Command

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 MVS.

- 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 MVS.

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.
- You need 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 mastername]
```

or

```
fileid AS mastername ON server READ
```

or

```
fileid LOCAL
```

or

```
fileid ON server
.
.
.
ind {WITH|INDEX} mastername
END
```

where:

*action*

Is one of the following:

**ADD** appends one or more new 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.

| For this platform... | The file ID is...                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------|
| For CMS              | Any valid file name, in <i>filename filetype filemode</i> format containing your FOCUS data source. |
| For MVS              | A <i>ddname</i> allocated to the MVS 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 MVS.

*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 MVS, 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 MVS) 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 the USE Directory**

To erase the USE directory, enter the following command:

```
USE CLEAR
```

## Specifying a Non-Default File ID

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 MVS:      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 FOCUS on the A-disk, described by the Master File EMPLOYEE, enter this USE command:

```
USE
EMP026 FOCUS A AS EMPLOYEE
END
```

After entering the USE command, you can read the EMP026 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

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 MVS, you must allocate the data source, with the MVS 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.

*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 MVS:      USE  
              WAGES NEW  
              END  
              CREATE FILE WAGES
```

## Accessing Data Sources 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 MVS:

```
USE  
EMPLOYEE READ  
END
```

# Concatenating 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 MVS:      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 MVS:      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 having 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 MVS:      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.

You can 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 MVS:

```
USE
EMP01 AS EMPLOYEE
ED01 AS EDUCFILE
EMP02 AS EMPLOYEE
ED02 AS EDUCFILE
END
```



## Accessing Simultaneous Usage Data Sources

In CMS, the FOCUS Database Server is a disconnected virtual machine that manages all READ/WRITE operations to a FOCUS data source.

In MVS, 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 CMS, is the file name, file type, and file mode of the FOCUS data source accessed by the disconnected virtual machine (FOCUS Database Server).

In MVS, is the ddname of the FOCUS data source allocated in the batch job or started task.

### *Example*

### Accessing an SU Data Source in CMS

If you want 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 MVS

If you want to use the EMPLOYEE FOCUS data source allocated to the FOCUS Database Server batch job or started task, two things must be done:

1. You must allocate a ddname to the communications data set that is allocated in the FOCUS Database Server batch job or started task pointing to the ddname FOCSU.

For example,

```
DYNAM ALLOC FILE MYSERVER DS prefix.FOCSU.DATA SHR
```

2. You must issue the USE command for your data source allocated in the batch job or started task.

```
USE  
EMPLOYEE ON MYSERVER  
END
```

## Multi-Thread Configuration

Performance gains may be achieved by routing READ only requests directly to the data source on disk instead of going through the FOCUS Database Server. This is called a multi-thread configuration. It is accomplished with the USE command and the keyword LOCAL.

### 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 MVS, 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.
- 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 that is named by the LOCATION attribute in the Master File, defaults to FOCUS in the local directory unless a USE command is issued. If the physical 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
```

---

## CHAPTER 10

# Providing Data Source Security: DBA

### Topics:

- Introduction
- Implementing Data Source Security
- Specifying Access Types: The ACCESS Attribute
- Limiting Data Source Access: The RESTRICT Attribute
- Placing Security Information in a Central Master File
- Hiding the Restriction Rules: The ENCRYPT Command
- FOCEXEC Security
- Program Accounting/Resource Limitation
- Absolute File Integrity

As Database Administrator, you can use FOCUS 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 (*Restricting Existing Files* on page 10-32) is not available for those data sources. Note that DBA security cannot protect a data source from non-FOCUS access.

## Introduction

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 10-9.
- You can restrict a user's access rights to read, write, or update only using the ACCESS attribute discussed in *Specifying Access Types: The ACCESS Attribute* on page 10-12.
- 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 10-17.
- 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 10-17.
- You can limit the values a user can 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 10-17.
- 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 10-25.
- You can use the FOCUSID exit routine to let an external security system set the FOCUS password.
- You can place security on FOCEXECs, which is discussed in *FOCEXEC Security* on page 10-34.

Program accounting, resource limitation, and the UACCT exit routine are discussed in *Program Accounting/Resource Limitation* on page 10-37.

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 10-40.

## Implementing 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) start following the END command in a Master File and tell FOCUS that security is needed for the data source and what type of security you want. Each security declaration can consist of one or several of the following attributes:

- The DBA attribute gives the name or password of the Database Administrator for the 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 to read data in a data source only.

W, which allows a user to write new segment instances to a data source only.

U, which allows a user to update records in a data source only.

- The RESTRICT attribute specifies certain segments or fields to which the user is not granted access. It can also be used to restrict the data values a user can see or perform transactions on.
- The NAME and VALUE attributes are part of the RESTRICT declaration.

You describe your 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

The following is a Master File that uses FOCUS security features:

```

FILENAME = PERS, SUFFIX = FOC,$
SEGMENT = IDSEG, SEGTYPE = S1,$
  FIELD = SSN           ,ALIAS = SSN      ,FORMAT = A9  ,$
  FIELD = FULLNAME     ,ALIAS = FNAME   ,FORMAT = A40 ,$
  FIELD = DIVISION     ,ALIAS = DIV     ,FORMAT = A8  ,$
SEGMENT=COMPSEG, PARENT=IDSEG, SEGTYPE=S1,$
  FIELD = SALARY       ,ALIAS = SAL     ,FORMAT = D8  ,$
  FIELD = DATE         ,ALIAS = DATE    ,FORMAT = YMD ,$
  FIELD = INCREASE     ,ALIAS = INC     ,FORMAT = D6  ,$
END
DBA=JONES76,$
USER=TOM ,ACCESS=RW, $
USER=BILL ,ACCESS=R ,RESTRICT=SEGMENT ,NAME=COMPSEG ,,$
USER=JOHN ,ACCESS=R ,RESTRICT=FIELD ,NAME=SALARY ,,$
                                     NAME=INCREASE ,,$
USER=LARRY ,ACCESS=U ,RESTRICT=FIELD ,NAME=SALARY ,,$
USER=TONY ,ACCESS=R ,RESTRICT=VALUE ,NAME=IDSEG,
  VALUE=DIVISION EQ 'WEST' ,,$
USER=MARY ,ACCESS=W ,RESTRICT=VALUE ,NAME=SALTEST,
  VALUE=INCREASE+SALARY GE SALARY,$
                                     NAME=HISTTEST,
  VALUE=DIV NE ' ' AND DATE GT 0,$

```

## Reference      Special Considerations

When using the JOIN command, it is possible to bypass the DBA information in a FOCUS 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 10-25. All data sources in the joined structure will get security information as coded in the DBAFILE.

## Identifying the DBA: The DBA Attribute

The first security attribute should be a password that identifies the Database Administrator. This password can be up to eight characters long. Since nothing else is needed, this line is terminated by the usual delimiter (,\$).

**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      Changing an Existing DBA Password

The DBA has the freedom to change any of the security attributes. If you change the DBA password in the Master File, you must use the RESTRICT command for existing data sources at the FOCUS command level (discussed in *Restricting Existing Files* on page 10-32) to inform each FOCUS data source affected by the change. Unless this is done, FOCUS will assume that the new description is an attempt to bypass the restriction rules. You use the following procedure for each data source affected:

1. Edit the Master File, changing the DBA value from old to new.

2. Issue the command:

```
SET PASS=old_DBA_value
```

3. Issue the command:

```
RESTRICT  
filename  
END
```

4. Issue the command:

```
SET PASS=new_value
```

**Note:** See the *Overview and Operating Environments* manual for specific syntax of the RESTRICT command for your operating environment.

## Including the DBA Attribute in HOLD Files

With the SET HOLDSTAT command (described in the *Developing Applications* manual), you can identify a data source containing DBA information and comments to be automatically included in HOLD and PCHOLD Master Files.

For MVS, 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 can system variables.

All lines from the HOLDSTAT file that appear prior to \$BOTTOM are placed at the top of the HOLD Master File, before any file and field declarations. All lines that appear after \$BOTTOM are appended to the bottom of the HOLD Master File. Any Dialogue Manager variables are replaced with the actual variable values.

### Example

### Including Comments 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 1999/05/20 at 17.58.10 by FOCUS 7.0   $
$           Database records retrieved=           19           $
$           Records in the HOLD file =           19           $
$=====
FILE = HOLD      , SUFFIX = FIX
SEGNAME = HOLD, SEGTYPE = S01
FIELDNAME = EMP_ID           ,E01           ,A9           ,A12           , $
FIELDNAME = LAST_NAME        ,E02           ,A15           ,A16           , $
FIELDNAME = FIRST_NAME       ,E03           ,A10           ,A12           , $
FIELDNAME = SALARY           ,E04           ,D12.2M       ,D08           , $

```

## 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 1999/05/20 at 17.58.10 by FOCUS 7.0 $
$           Database records retrieved=          19         $
$           Records in the HOLD file =          19         $
$=====
FILE = HOLD ,SUFFIX = FIX
SEGNAME = HOLD, SEGTYPE = S01
FIELDNAME = EMP_ID           ,E01           ,A9           ,A12           , $
FIELDNAME = LAST_NAME       ,E02           ,A15           ,A16           , $
FIELDNAME = FIRST_NAME     ,E03           ,A10           ,A12           , $
FIELDNAME = SALARY         ,E04           ,D12.2M       ,D08           , $
END
DBA=MARY,$
    
```

## Identifying Users With Access Rights: 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 Access Types: The ACCESS Attribute* on page 10-12) to specify what sort of ACCESS the user is granted.

Before using a secured data source, a user must enter his or her password using the SET PASS command. If that password is not included in the Master File, the user is denied access to the 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 is displayed:

```
(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 eight characters for the user.

For example:

```
USER=TOM,...
```

You can specify a blank password. Such a password does not require the user to issue a SET PASS= command. A blank password may still have access limits and is convenient when a number of users have the same access rights. An example of setting a user's password to blank, and access to read only follows:

```
USER= , ACCESS=R,$
```

## Establishing User Identity

A user must enter his or her password before using any FOCUS 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.

*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
```

FOCUS maintains a list of the files for which a user has set specific passwords. To see the list of files, issue:

```
? PASS
```

When the user sets a password IN \* (all files), the list of active passwords collapses to one entry with no associated file name. To retain the file name list, use the NOCLEAR option.

In the next example, the password KEN replaces all passwords active in all files, and the table of active passwords is folded to one entry:

```
SET PASS=KEN IN *
```

In the following, MARY replaces all passwords in the existing table of active passwords (which consists of files NINE and TEN) but FRANK is the password for all other files. The option NOCLEAR provides a shorthand way to replace all passwords in a specific list:

```
SET PASS=BILL IN NINE,TOM IN TEN  
SET PASS=MARY IN * NOCLEAR,FRANK
```

**Note:** The FIND function does not work with COMBINED data sources secured with different passwords.

Users must issue their passwords using the SET PASS command during each FOCUS session in which they use a secured data source. They may issue their passwords at any time before using the data source and can issue a different password afterward to access another data source.

## Specifying Access Types: The ACCESS Attribute

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 FOCUS commands a user can issue. Before you decide what access levels to assign to a user, you must consider what commands that user will need. If a user does not have sufficient access rights to use a given command, the following error message will be displayed:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE:  
filename
```

ACCESS levels determine what a user can do to the data source. You use the RESTRICT attribute (discussed in *Limiting Data Source Access: The RESTRICT Attribute* on page 10-17) 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 he or she is granted.

**Types of Access**

| <b>Command</b>  | <b>R</b> | <b>W</b> | <b>RW</b> | <b>U</b> | <b>DBA</b> |
|-----------------|----------|----------|-----------|----------|------------|
| <b>CHECK</b>    | X        | X        | X         | X        | X          |
| <b>CREATE</b>   |          |          | X         |          | X          |
| <b>DECRYPT</b>  |          |          |           |          | X          |
| <b>DEFINE</b>   | X        |          | X         |          | X          |
| <b>ENCRYPT</b>  |          |          |           |          | X          |
| <b>FSCAN</b>    |          | X        | X         | X        | X          |
| <b>HLI</b>      |          |          | X         |          | X          |
| <b>MAINTAIN</b> |          | X        | X         | X        | X          |
| <b>MATCH</b>    | X        |          | X         |          | X          |
| <b>MODIFY</b>   |          | X        | X         | X        | X          |
| <b>REBUILD</b>  |          |          | X         |          | X          |
| <b>RESTRICT</b> |          |          |           |          | X          |
| <b>SCAN</b>     |          |          | X         | X        | X          |
| <b>TABLE</b>    | X        |          | X         |          | X          |

### The CHECK Command

Users without the DBA password or read/write access are allowed limited access to the CHECK command. However, when the HOLD option is specified, the warning ACCESS LIMITED BY PASSWORD is produced, and restricted fields are propagated to the HOLD file depending on the DBA RESTRICT attribute. Refer to *Limiting Data Source Access: The RESTRICT Attribute* on page 10-17 for more information on the RESTRICT attribute.



## Reference

### RESTRICT Attribute Keywords

The RESTRICT attribute keywords affect the resulting HOLD file as follows:

#### FIELD

Fields named with the NAME parameter are not included in the HOLD file.

#### SEGMENT

The segments named with the NAME parameter are included, but fields in those segments are not.

#### SAME

The behavior is the same as for the user named in the NAME parameter.

#### NOPRINT

Fields named in the NAME or SEGNAME parameter are included since 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.

## The CREATE Command

Only users with the DBA password or read/write (RW) access rights can issue a CREATE command.

## The DECRYPT Command

Only users with the DBA password can issue a DECRYPT command.

## The DEFINE Command

As with all reporting commands, a user need only have an access of R (read only) to use the DEFINE command. An access of R permits the user to read records from the 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).

## The ENCRYPT Command

Only users with the DBA password can use the ENCRYPT command.

## Host Language Interface (HLI)

In order to have use of the Host Language Interface, a user must have read/write (RW) access. With ACCESS=RW, FIELD and SEGMENT restrictions are active, but VALUE restrictions are not. (See *Limiting Data Source Access: The RESTRICT Attribute* on page 10-17 for information on these restrictions.)

The password is placed in the File Control Block (FCB), words 19 and 29 (byte 73 to 80).

## The MODIFY 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 cannot use the MODIFY or MAINTAIN command.

## **The REBUILD Command**

Only users with the DBA password or read/write (RW) access rights can issue the REBUILD command.

## **The RESTRICT Command**

Only users with the DBA password may use the RESTRICT command.

## **The FSCAN Facility**

Users with ACCESS=RW have unlimited access to the 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 displays 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.

## **The 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.

## **The 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

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 10-37).

SAME

Specifies that the user has the same restrictions as the user named in the NAME parameter. No more than four nested SAME users are valid.

NOPRINT

Specifies that the field named in the NAME or SEGMENT parameter can be mentioned in a request statement, but will not be displayed.

*name*

Is the name of the field or segment you wish to restrict. When used after NOPRINT, this can only be a field name. NAME=SYSTEM, which can only be used with value tests, restricts every segment in the 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 Fields and Segments

The RESTRICT attribute identifies the segments or fields that the user will not be able to access. Anything not named in the RESTRICT attribute will be accessible.

Without the RESTRICT attribute, the user has access to the entire 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 Fields and Segments

The syntax to restrict access to a field or segment is

```
... RESTRICT=level, NAME=name, $
```

where:

*level*

Can be one of the following:

**FIELD**

Specifies that the 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 be displayed. When used after NOPRINT, NAME can only be a field name.

*name*

Is the name of the field or segment you wish to restrict. When used after NOPRINT, this can only be a field name.

NAME=SYSTEM, which can only be used with value tests, restricts every segment in the 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 be displayed. The data will appear as blanks for alphanumeric format or zeros for numeric fields.
- You can restrict multiple fields or segments by providing multiple RESTRICT statements. For example, if you wish to restrict Harry from using both field A and segment B, you issue the following access limits:

```
USER=HARRY, ACCESS=R,          RESTRICT=FIELD , NAME=A,$  
                                RESTRICT=SEGMENT, NAME=B,$
```

- You can restrict as many segments and fields as you like.
- Using RESTRICT=SAME is a convenient way to reuse a common set of restrictions for more than one password. If you specify RESTRICT=SAME and provide a user name or password as it is specified in the USER attribute for the NAME value, the new user will be subject to the same restrictions as the one named in the NAME attribute. You can then add additional restrictions, as they are needed.

*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 Values

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: you can restrict the values the user can read from the data source, or you can restrict 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.

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 can access all data values in the data source. The user will 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 10-32. 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 on which you are performing the tests. 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, however, the test is made effective for requests on its parent, IDSEG. In this case, the request PRINT FULLNAME would only print the full names of people who meet this test, that is, whose salary is less than or equal to \$10,000, even though the test is performed on a field that is part of a descendant segment of IDSEG. If, however, the test was made effective on COMPSEG, that is, NAME=COMPSEG, then the full name of everyone in the data source could be retrieved, but with the salary information of only those meeting the test condition.



## Restricting Values a User Can Write

If a user's access rights are either W or U, VALUE tests used with the MODIFY command validate new transactions. The format of the test conditions are those used in the ON MATCH VALIDATE expressions of the MODIFY command, which is discussed in the *Maintaining Databases* manual.

There are two different ways you can restrict the values a user can write to a data source: you can restrict the values the user actually is allowed to enter, or you can restrict the values that the user is allowed to change. You must supply an ACCESS=R restriction to restrict the user from seeing certain data values in the 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. Since it does not use the data source, 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, you may want to restrict the values a user can enter on a segment level by supplying a segment name for NAME=. This creates a condition similar to an ON MATCH VALIDATE phrase.

```
...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 right before the action (UPDATE or DELETE) and can reference D. fields.

**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 he can enter for the DIVISION field is EAST.

The segment name on which the test is to be applied is given as the NAME parameter. If a request statement does not perform any action on this segment, the test itself is not performed. This is true even if you are making changes to a segment that is a child of the segment on which the test is performed.

The VALUE tests are added to any VALIDATE conditions that the MODIFY request contains. Only transactions passing both the VALIDATE and VALUE tests are accepted for processing.

## Restricting Both Read and Write Values

In many cases it will prove useful to issue both ACCESS=W (for MODIFY) and ACCESS=R (for TABLE) value restrictions for a user. This will both limit the values a user can write to the data source and limit the data values that the user can actually see. You 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

```
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

The DBAFILE attribute enables you to place all of the passwords and restrictions for many Master Files in one central file. Each individual Master File points to this central control file. Groups of Master Files with the same DBA password may share a common DBAFILE which itself has the same DBA password.

There are several benefits to this technique. The primary ones are:

- Passwords only have to be stored once when they are applicable to a group of data sources. This simplifies password administration.
- Data sources with different DBA passwords can now 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 may include additional attributes before the END statement that signifies the presence of DBA information. The DBA password in the DBAFILE is the same as the password in all the Master Files that refer to it. This prevents individuals from substituting their own security. All of these Master Files should be encrypted.
- 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 above, 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). Consult *File Naming Requirements for DBAFILE* on page 10-27 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=dbname, DBAFILE=filename , $
```

where:

*dbname*

Is the same as the dbname 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=ffff, $
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 not known by the DBAFILE.

### *Example*

### DBAFILE Naming Conventions

```

ONE MASTER
FILENAME=XONE
.
.
.
END
DBA=ABC, DBAFILE=FOUR, $

```

```

FOUR MASTER
FILENAME=FOUR
.
.
.
END
DBA=ABC, $
.
.
.
FILENAME=XONE, $
.
.
.

```

ONE MASTER is referred to in requests as TABLE FILE ONE. However, both ONE MASTER and the DBA section of the DBAFILE, FOUR MASTER, specify FILENAME=XONE.

## Connection to Existing DBA System 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.

Once you start to use the new system, you should convert all of your Master Files. For database administrators who want to convert existing systems but do not want a separate physical DBAFILE, the DBAFILE attribute can specify the 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 now JOIN and COMBINE data sources that come from different applications and have different DBA 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.

## Using Filters

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.<br>Values are:<br>R read only<br>W write new segments only<br>RW read and write<br>U update values only |
| RESTRICT  | RESTRICT | 8              | Types of restrictions to be imposed for this access level.<br>Values are:<br>SEGMENT<br>FIELD<br>VALUE<br>SAME<br>PROGRAM<br>NOPRINT    |
| NAME      | NAME     | 66             | Name of segment or field restricted or of the program to be called.                                                                     |
| VALUE     | VALUE    | 80             | Test expression which must be true when RESTRICT=VALUE is the type of limit.                                                            |
| DBAFILE   | DBAFILE  | 8              | Names the Master File that contains passwords and restrictions to use.                                                                  |



## Hiding the Restriction Rules: The ENCRYPT Command

Since the restriction information for a FOCUS data source is stored in its Master File, you will want to encrypt the Master File in order to prevent users from examining the restriction rules. Only the Database Administrator can encrypt a description. Thus, you must set PASS=DBAname before you issue the ENCRYPT command. The syntax of the ENCRYPT command varies from operating system to operating system. See the *Overview and Operating Environments* manual for information on your operating system.

### *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 Master Files

The following is an example of the complete procedure:

```
SET PASS=JONES76  
ENCRYPT FILE PERS
```

The process can be reversed if you wish to change the restrictions. The command to restore the description to a readable form is DECRYPT.

The DBA password must be issued with the SET command before the file can be decrypted. For example:

```
SET PASS=JONES76  
DECRYPT FILE PERS
```

## Encrypting Data

You may also use the ENCRYPT command within the Master File to encrypt some or all of its segments. When encrypted files are stored on their external media (disk or tape) they are secure from unauthorized examination.

Encryption takes place on the segment level; that is, the entire segment is encrypted. The request for encryption is made in the Master File by setting the attribute ENCRYPT to ON.

### Example

#### Encrypting Data

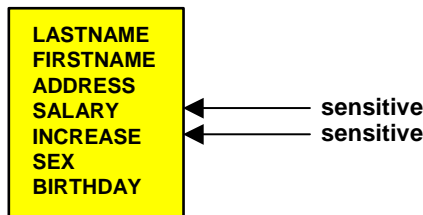
```
SEGMENT=COMPSEG, PARENT=IDSEG, SEGTYPE=S1, ENCRYPT=ON,$
```

You must specify the ENCRYPT attribute before you enter any data in the data source. The message NEW FILE... must appear when the encryption is first requested.

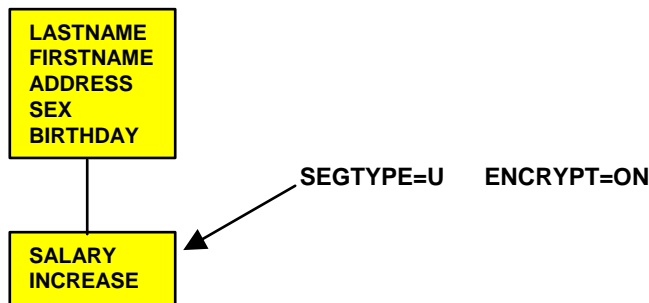
Encryption cannot be requested later by a change to the Master File and cannot be removed once it has been requested and any data has been entered in the data source.

## Performance Considerations for Encrypted Data

There is a small loss in processing efficiency when data is encrypted. You can minimize this loss by grouping the sensitive data fields together on a segment and making them a separate segment of SEGTYPE=U, unique segment, beneath their original segment. For example, suppose the data items on a segment are:



They should be grouped as:



## Restricting Existing Files

When you write a new Master File for a new data source and include security limitations, data added to the data source is automatically protected according to those rules. If you write a new Master File for an already existing data source that contains no data (that is, one in which the FOCUS MODIFY or MAINTAIN command produces the message NEW FILE or one for which a CREATE FILE command must be issued in MVS) the data will also be automatically protected. If, however, you have existing files to which you want to add security limitations, you need to use the RESTRICT command. (**Note:** This is not the RESTRICT attribute described in *Limiting Data Source Access: The RESTRICT Attribute* on page 10-17.)

If an existing FOCUS data source has no DBA rules and you want to add DBA rules, perform the following:

1. Edit the Master File and add the DBA rules.
2. Issue the RESTRICT command to write the DBA password to the data source.

**Note:** The RESTRICT command cannot be used for non-FOCUS data sources. See the *Overview and Operating Environments* manual for specific syntax of the RESTRICT command for your operating environment.

## Displaying the Decision Table

When you enter security attributes for a Master File, FOCUS creates an internal decision table that lists users and their access privileges. You can display the decision table associated with a given 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

Display 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:

```
>
ex dbatable
FILE NAME:
pers
>
>
```

PAGE 1

DATA BASE ADMINISTRATOR JONES76  
DBAFILE PERS ON 01/18/94 AT 13.09.07

| FILENAME | USER  | ACCESS | RESTRICT | NAME     | VALUE                     |
|----------|-------|--------|----------|----------|---------------------------|
| PERS     |       |        |          |          |                           |
|          | BILL  | R      | SEGMENT  | COMPSEG  |                           |
|          | JOHN  | R      | FIELD    | SALARY   |                           |
|          |       | R      | FIELD    | INCREASE |                           |
|          | LARRY | U      | FIELD    | SALARY   |                           |
|          | MARY  | W      | VALUE    | SALTEST  | INCREASE+SALARY GE SALARY |
|          |       | W      | VALUE    | HISTTEST | DIV NE ' ' AND DATE GT 0  |
|          | TOM   | RW     |          |          |                           |
|          | TONY  | R      | VALUE    | IDSEG    | DIVISION EQ 'WEST'        |

## Setting Passwords Externally

Passwords can also be set automatically by an external security system such as RACF<sup>®</sup>, CA-ACF2<sup>®</sup>, or CA-Top Secret<sup>®</sup>. Passwords issued this way are set when FOCUS is first entered and may be permanent (that is, not alterable by subsequent SET USER, SET PASS or -PASS commands); or they may be default passwords that can be subsequently overridden; or they may be permanent for some users, defaults for other users, and not set at all for yet other users.

The advantage of setting FOCUS passwords externally is that the password need not be known by the user, does not require prompting, and does not have to be embedded in a PROFILE FOCEXEC or an encrypted FOCEXEC.

Passwords set this way must match the passwords specified in the Master Files of the data sources being accessed.

See your installation documentation for FOCUSID installation instructions.

## FOCEXEC Security

Most data security issues are best handled by the FOCUS DBA facility. Nevertheless, some additional data security facilities are incorporated within Dialogue Manager. These are:

- Suppressing password display.
- Setting passwords in encrypted FOCEXECs.
- Defining variable passwords.
- Encrypting and decrypting FOCEXECs.
- Locking FOCEXEC users out of FOCUS.

External security systems can also set passwords through the FOCUSID exit routine.

## Suppressing Password Display

The NODISPLAY attribute can be used in combination with -CRTFORM and -PASS 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 Passwords in Encrypted FOCEXECs

Passwords can be set within FOCEXECs and tied to different portions of FOCEXECs according to this syntax:

```
-PASS password
```

Since -PASS is a Dialogue Manager 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 you edit the FOCEXEC. It does not appear when the ECHO option is ALL and is not printed in a batch run log.

## Encrypting and Decrypting FOCEXECs

You may want to keep the actual text of a stored FOCEXEC confidential while allowing users to execute the FOCEXEC. You may want to do this either because there is confidential information stored in the FOCEXEC or because you do not want the FOCEXEC changed by unauthorized users. You can protect a stored FOCEXEC from unauthorized users with the ENCRYPT command.

Any user can execute an encrypted FOCEXEC, but you must decrypt the FOCEXEC to view it. Only a user with the DBA password can decrypt the FOCEXEC.

### *Syntax*

### How to Encrypt and Decrypt FOCEXECs

You use the following procedure to encrypt the FOCEXEC named SALERPT:

```
SET PASS = DOHIDE  
ENCRYPT FILE SALERPT FOCEXEC
```

Anyone can execute the FOCEXEC by typing EX SALERPT. The FOCEXEC can only be viewed by decrypting it, as follows:

```
SET PASS = DOHIDE  
DECRYPT FILE SALERPT FOCEXEC
```

Encrypted FOCEXECs cannot be echoed (that is, have their commands displayed on the terminal), so &ECHO has no effect.

## Locking FOCEXEC Users Out of FOCUS

Users can normally respond to a Dialogue Manager value request with QUIT and return to the FOCUS command level. In situations where it is important to prevent users from entering or returning to FOCUS, the environment can be locked and QUIT can be deactivated by entering in a FOCEXEC:

```
-SET &QUIT=OFF;
```

With QUIT deactivated, any attempt to leave the Dialogue Manager environment will produce an error message. Following the error message, the user will be reprompted for the needed value.

A user may still terminate the session from inside a locked environment by responding to a prompt with:

```
QUIT FOCUS
```

This returns the user to the operating system, not to the FOCUS command level.

The default setting for &QUIT is ON.

## Program Accounting/Resource Limitation

In addition to controlling access to a 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 people's 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

You can use FOCUS security attributes to specify that a user-written program be called immediately after a `TABLE` or `GRAPH` command has completed, but before the report is printed. You activate this user-written program by assigning the value `PROGRAM` to the `RESTRICT` attribute. The program is passed the statistics of the run (that is, number of records retrieved, lines of sorted results) and the identity of each data field that was active in the run. You can use this user program exit to:

- Monitor the retrieval activity for particular data sources. For instance: number of requests, number of records retrieved, distribution of usage by user category (by userid or password identity).
- Monitor the usage frequency of items in a 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

You activate a DBA user program by adding the following attributes to the security section of any Master File for each user that you want the program to monitor

```
USER=name, ACCESS=R, RESTRICT=PROGRAM,  
NAME=pgmname, VALUE=returncode, $
```

where:

*name*

Is the arbitrary code name used to identify the user (8 bytes).

*pgmname*

Is the name of the user-written program (8 bytes).

*returncode*

Must be matched with the DBA name (8 bytes).

For example:

```
USER=PETER, ACCESS=R, RESTRICT=PROGRAM,  
NAME=PETER1, VALUE=D76, $
```

You can specify other restrictions for the users mentioned in addition to calling the program.

## Specifications for the User-Written Program

The user program must be coded as a subroutine in a language 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 of these are computed by FOCUS, the last is returned by the user program to FOCUS. The value of the last argument is matched to a value provided in the DBA section of the Master File. The purpose of this is to prevent a spurious program of the same name from being substituted for the real one. If the DBA value and the retrieval value do not match, the report is not printed and FOCUS exits immediately.

The arguments to the call are:

| Argument | Format     | Length       | Description                                                                                           |
|----------|------------|--------------|-------------------------------------------------------------------------------------------------------|
| FILEID   | Alpha      | 18 bytes     | The name of the 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 will limit 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

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, change 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.

## 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 command

```
SET SHADOW = value
```

where:

*value*

Can be one of the following:

**OFF**

Does not invoke Absolute File Integrity. This is the default value.

**ON**

Invokes Absolute File Integrity.

**OLD**

Invokes the use of the shadow technology available in FOCUS releases prior to 7.0 (before the size limit for FOCUS data sources was increased). This means 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 Existing Data Sources

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.

---

## APPENDIX A

# Master Files and Diagrams

### Topics:

- Creating Sample Data Sources
- The EMPLOYEE Data Source
- The JOBFIL Data Source
- The EDUCFIL Data Source
- The SALES Data Source
- The PROD Data Source
- The CAR Data Source
- The LEDGER Data Source
- The FINANCE Data Source
- The REGION Data Source
- The COURSES Data Source
- The EMPDATA Data Source
- The EXPERSO Data Source
- The TRAINING Data Source
- The PAYHIST File
- The COMASTER File
- The VideoTrk and MOVIES Data Sources
- The VIDEOTR2 Data Source
- The Gotham Grinds Data Sources

This appendix contains data source descriptions and structure diagrams for the examples used throughout the documentation.

## Creating Sample Data Sources

You can create the 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,<br>EDUCFILE, and<br>JOBFILE              | Under CMS enter:<br><br><code>EX EMPTEST</code><br><br>Under MVS, enter:<br><br><code>EX EMPTSO</code><br><br>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 will replace the data sources with new copies. This FOCEXEC assumes that the high-level qualifier for the FOCUS data sources will be the same as the high-level qualifier for the MASTER PDS that was unloaded from the tape. |
| SALES<br>PROD                                      | <code>EX SALES</code><br><code>EX PROD</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| CAR                                                | none (created automatically during installation)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| LEDGER<br>FINANCE<br>REGION<br>COURSES<br>EXPERSON | <code>EX LEDGER</code><br><code>EX FINANCE</code><br><code>EX REGION</code><br><code>EX COURSES</code><br><code>EX EXPERSON</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| EMPDATA<br>TRAINING                                | <code>EX LOADEMP</code><br><code>EX LOADTRAI</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 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 and<br>MOVIES                             | <code>EX LOADVTRK</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| VIDEOTR2                                           | <code>EX LOADVID2</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Gotham Grinds                                      | <code>EX LOADGG</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## The EMPLOYEE Data Source

The EMPLOYEE data source contains data about a company's employees. Its segments are:

- EMPINFO, which contains employee IDs, names, and positions.
- FUNDTRAN, which specifies employees' direct deposit accounts. This segment is unique.
- PAYINFO, which contains the employee's salary history.
- ADDRESS, which contains employees' home and bank addresses.
- SALINFO, which contains data on employees' monthly pay.
- DEDUCT, which contains data on monthly pay deductions.

The EMPLOYEE data source also contains cross-referenced segments belonging to the JOBFIL and EDUCFIL files, described later in this appendix. The segments are:

- JOBSEG (from JOBFIL), which describes the job positions held by each employee.
- SECSEG (from JOBFIL), which lists the skills required by each position.
- SKILLSEG (from JOBFIL), which specifies the security clearance needed for each job position.
- ATTNDSEG (from EDUCFIL), which lists the dates that employees attended in-house courses.
- COURSEG (from EDUCFIL), which lists the courses that the employees attended.

## The EMPLOYEE Master File

```

FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMP INFO, 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=EMP INFO
  FIELDNAME=BANK_NAME, ALIAS=BN, FORMAT=A20, $
  FIELDNAME=BANK_CODE, ALIAS=BC, FORMAT=I6S, $
  FIELDNAME=BANK_ACCT, ALIAS=BA, FORMAT=I9S, $$
  FIELDNAME=EFFECT_DATE, ALIAS=EDATE, FORMAT=I6YMD, $
SEGNAME=PAYINFO, SEGTYPE=SH1, PARENT=EMP INFO
  FIELDNAME=DAT_INC, ALIAS=DI, FORMAT=I6YMD, $
  FIELDNAME=PCT_INC, ALIAS=PI, FORMAT=F6.2, $
  FIELDNAME=SALARY, ALIAS=SAL, FORMAT=D12.2M, $
  FIELDNAME=JOBCODE, ALIAS=JBC, FORMAT=A3, $
SEGNAME=ADDRESS, SEGTYPE=S1, PARENT=EMP INFO
  FIELDNAME=TYPE, ALIAS=AT, FORMAT=A4, $
  FIELDNAME=ADDRESS_LN1, ALIAS=LN1, FORMAT=A20, $$
  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=EMP INFO
  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=EMP INFO, CRFILE=EDUCFILE, CRKEY=EMP_ID,$
SEGNAME=COURSEG, SEGTYPE=KLU, PARENT=ATTNDSEG, CRFILE=EDUCFILE,$

```



# The EMPLOYEE Structure Diagram

STRUCTURE OF FOCUS FILE EMPLOYEE ON 09/15/00 AT 10.16.27

```

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

```

## The JOBFIL Data Source

The JOBFIL data source contains information on a company's job positions. Its segments are:

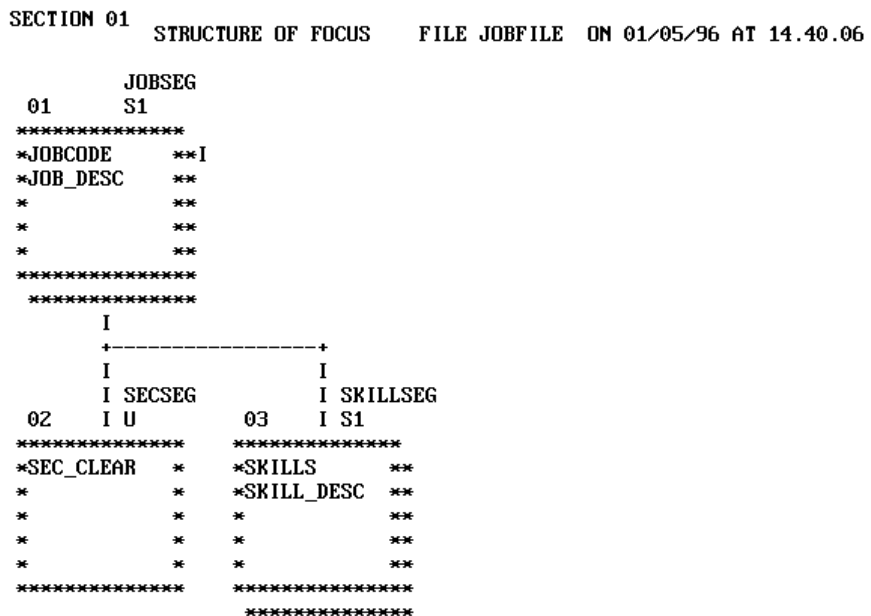
- JOBSEG describes what each position is. The field JOBCODE in this segment is indexed.
- SKILLSEG lists the skills required by each position.
- SECSEG specifies the security clearance needed, if any. This segment is unique.

## The JOBFIL Master File

```

FILENAME=JOBFIL ,SUFFIX=FOC
SEGNAME=JOBSEG ,SEGTYPE=S1
FIELD=JOBCODE ,ALIAS=JC ,USAGE=A3 ,INDEX=I,$
FIELD=JOB_DESC ,ALIAS=JD ,USAGE=A25 ,,$
SEGNAME=SKILLSEG ,SEGTYPE=S1 ,PARENT=JOBSEG
FIELD=SKILLS ,ALIAS= ,USAGE=A4 ,,$
FIELD=SKILL_DESC ,ALIAS=SD ,USAGE=A30 ,,$
SEGNAME=SECSEG ,SEGTYPE=U ,PARENT=JOBSEG
FIELD=SEC_CLEAR ,ALIAS=SC ,USAGE=A6 ,,$
    
```

## The JOBFIL Structure Diagram



## The EDUCFILE Data Source

The EDUCFILE data source contains data on a company's in-house courses. Its segments are:

- COURSESEG contains data on each course.
- ATTNDSEG specifies which employees attended the courses. Both fields in the segment are key fields. The field EMP\_ID in this segment is indexed.

## The EDUCFILE Master File

```

FILENAME=EDUCFILE ,SUFFIX=FOC
SEGNAME=COURSESEG ,SEGTYPE=S1
  FIELD=COURSE_CODE ,ALIAS=CC           ,USAGE=A6           ,$
  FIELD=COURSE_NAME ,ALIAS=CD           ,USAGE=A30          ,$
SEGNAME=ATTNDSEG ,SEGTYPE=SH2 ,PARENT=COURSESEG
  FIELD=DATE_ATTEND ,ALIAS=DA           ,USAGE=I6YMD        ,$
  FIELD=EMP_ID      ,ALIAS=EID          ,USAGE=A9           ,INDEX=I,$

```

## The EDUCFILE Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS   FILE EDUCFILE ON 01/05/96 AT 14.45.44
          COURSESEG
01      S1
*****
*COURSE_CODE **
*COURSE_NAME **
*           **
*           **
*           **
*****
          I
          I
          I
          I ATTNDSEG
02      I SH2
*****
*DATE_ATTEND **
*EMP_ID      **I
*           **
*           **
*           **
*****
*****

```

## The SALES Data Source

The SALES data source records sales data for a dairy company (or a store chain). Its segments are:

- STOR\_SEG lists the stores buying the products.
- DAT\_SEG contains the dates of inventory.
- PRODUCT contains sales data for each product on each date. Note the following about fields in this segment:
  - The PROD\_CODE field is indexed.
  - The RETURNS and DAMAGED fields have the MISSING=ON attribute.

## The SALES Master File

```
FILENAME=KSALES, SUFFIX=FOC,
SEGNAME=STOR_SEG, SEGTYPE=S1,
  FIELDNAME=STORE_CODE, ALIAS=SNO, FORMAT=A3, $
  FIELDNAME=CITY, ALIAS=CTY, FORMAT=A15, $
  FIELDNAME=AREA, ALIAS=LOC, FORMAT=A1, $
SEGNAME=DATE_SEG, PARENT=STOR_SEG, SEGTYPE=SH1,
  FIELDNAME=DATE, ALIAS=DTE, FORMAT=A4MD, $
SEGNAME=PRODUCT, PARENT=DATE_SEG, SEGTYPE=S1,
  FIELDNAME=PROD_CODE, ALIAS=PCODE, FORMAT=A3, FIELDTYPE=1, $
  FIELDNAME=UNIT_SOLD, ALIAS=SOLD, FORMAT=I5, $
  FIELDNAME=RETAIL_PRICE, ALIAS=RP, FORMAT=D5.2M, $
  FIELDNAME=DELIVER_AMT, ALIAS=SHIP, FORMAT=I5, $
  FIELDNAME=OPENING_AMT, ALIAS=INV, FORMAT=I5, $
  FIELDNAME=RETURNS, ALIAS=RTN, FORMAT=I3, MISSING=ON, $
  FIELDNAME=DAMAGED, ALIAS=BAD, FORMAT=I3, MISSING=ON, $
```

# The SALES Structure Diagram

SECTION 01

STRUCTURE OF FOCUS

FILE SALES ON 01/05/96 AT 14.50.28

```

          STOR_SEG
01      S1
*****
*STORE_CODE **
*CITY      **
*AREA     **
*         **
*         **
*****
          I
          I
          I
          I DATE_SEG
02      I SH1
*****
*DATE     **
*         **I
*         **
*         **
*         **
*****
          I
          I
          I
          I PRODUCT
03      I S1
*****
*PROD_CODE **I
*UNIT_SOLD **
*RETAIL_PRICE**
*DELIVER_AMT **
*         **
*****
          I
          I
          I
          I

```

## The 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.

## The PROD Master File

```
FILE=KPROD, SUFFIX=FOC,  
SEGMENT=PRODUCT, SEGTYPE=S1,  
  FIELDNAME=PROD_CODE, ALIAS=PCODE, FORMAT=A3, FIELDTYPE=I, $  
  FIELDNAME=PROD_NAME, ALIAS=ITEM,  FORMAT=A15, $  
  FIELDNAME=PACKAGE, ALIAS=SIZE,  FORMAT=A12, $  
  FIELDNAME=UNIT_COST, ALIAS=COST,  FORMAT=D5.2M, $
```

## The PROD Structure Diagram

```
SECTION 01  
          STRUCTURE OF FOCUS   FILE PROD   ON 01/05/94 AT 14.57.38  
  
          PRODUCT  
01      S1  
*****  
*PROD_CODE **I  
*PROD_NAME **  
*PACKAGE **  
*UNIT_COST **  
* **  
*****  
*****
```

## The CAR Data Source

The CAR data source contains specifications and sales information for rare cars. Its segments are:

- ORIGIN lists the country that manufactures the car. The field COUNTRY is indexed.
- COMP contains the car name.
- CARREC contains the car model.
- BODY lists the body type, seats, dealer and retail costs, and units sold.
- SPECS lists car specifications. This segment is unique.
- WARRANT lists the type of warranty.
- EQUIP lists standard equipment.

The aliases in the CAR Master File are specified without the ALIAS keyword.

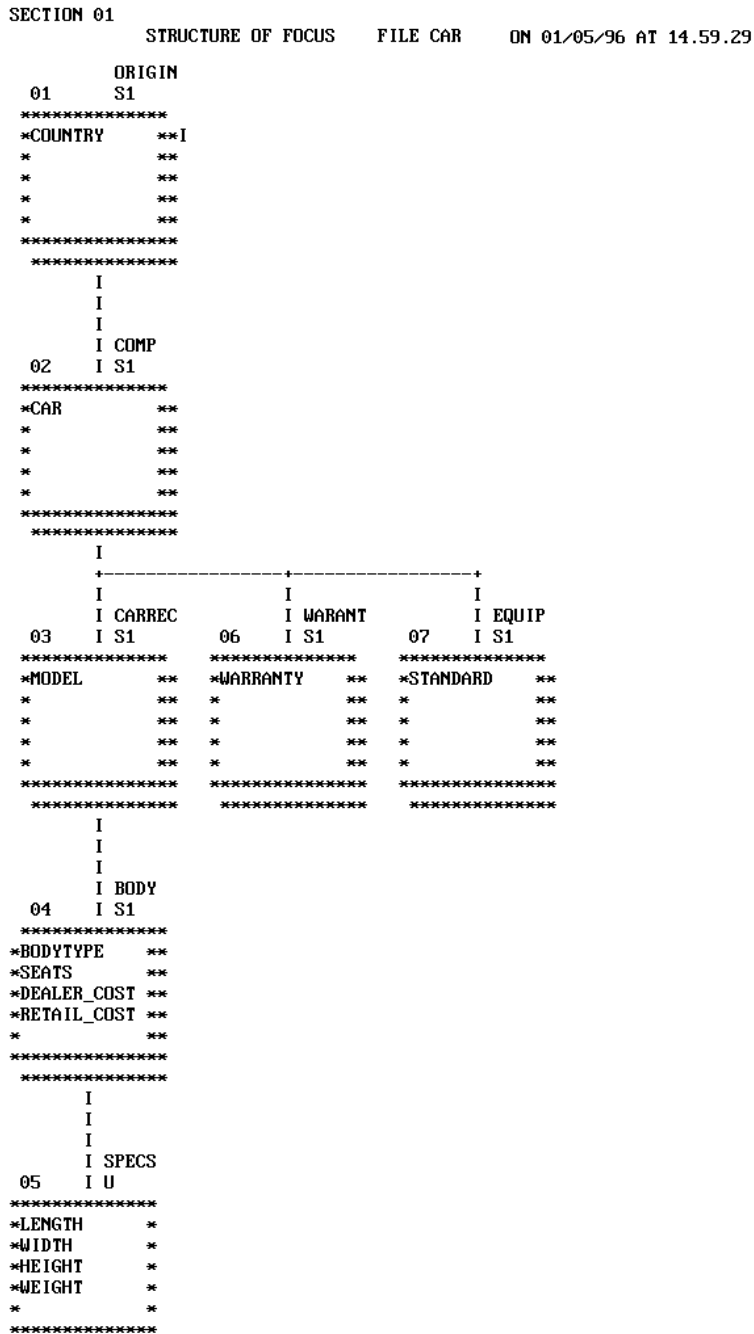
## The CAR Master File

```

FILENAME=CAR,SUFFIX=FOC
SEGNAME=ORIGIN,SEGTYPE=S1
  FIELDNAME=COUNTRY,COUNTRY,A10,FIELDTYPE=I,$
SEGNAME=COMP,SEGTYPE=S1,PARENT=ORIGIN
  FIELDNAME=CAR,CARS,A16,$
SEGNAME=CARREC,SEGTYPE=S1,PARENT=COMP
  FIELDNAME=MODEL,MODEL,A24,$
SEGNAME=BODY,SEGTYPE=S1,PARENT=CARREC
  FIELDNAME=BODYTYPE,TYPE,A12,$
  FIELDNAME=SEATS,SEAT,I3,$
  FIELDNAME=DEALER_COST,DCOST,D7,$
  FIELDNAME=RETAIL_COST,RCOST,D7,$
  FIELDNAME=SALES,UNITS,I6,$
SEGNAME=SPECS,SEGTYPE=U,PARENT=BODY
  FIELDNAME=LENGTH,LEN,D5,$
  FIELDNAME=WIDTH,WIDTH,D5,$
  FIELDNAME=HEIGHT,HEIGHT,D5,$
  FIELDNAME=WEIGHT,WEIGHT,D6,$
  FIELDNAME=WHEELBASE,BASE,D6.1,$
  FIELDNAME=FUEL_CAP,FUEL,D6.1,$
  FIELDNAME=BHP,POWER,D6,$
  FIELDNAME=RPM,RPM,I5,$
  FIELDNAME=MPG,MILES,D6,$
  FIELDNAME=ACCEL,SECONDS,D6,$
SEGNAME=WARRANT,SEGTYPE=S1,PARENT=COMP
  FIELDNAME=WARRANTY,WARR,A40,$
SEGNAME=EQUIP,SEGTYPE=S1,PARENT=COMP
  FIELDNAME=STANDARD.EQUIP.A40.$

```

# The CAR Structure Diagram





# The LEDGER Data Source

The LEDGER data source lists accounting information. 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.

## The LEDGER Master File

```

FILENAME=LEDGER, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
  FIELDNAME=YEAR   , , FORMAT=A4, $
  FIELDNAME=ACCOUNT, , FORMAT=A4, $
  FIELDNAME=AMOUNT , , FORMAT=I5C,$
    
```

## The LEDGER Structure Diagram

```

SECTION 01          STRUCTURE OF FOCUS   FILE LEDGER   ON 01/05/96 AT 15.07.56

          TOP
01       S2
*****
*YEAR           **
*ACCOUNT        **
*AMOUNT         **
*               **
*               **
*****
*****
    
```

## The FINANCE Data Source

The FINANCE data source contains financial information 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.

## The FINANCE Master File

```
FILENAME=FINANCE, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
FIELDNAME=YEAR , , FORMAT=A4, $
FIELDNAME=ACCOUNT, , FORMAT=A4, $
FIELDNAME=AMOUNT , , FORMAT=D12C,$
```

## The FINANCE Structure Diagram

```
SECTION 01          STRUCTURE OF FOCUS      FILE FINANCE  ON 01/05/96 AT 15.17.08

          TOP
01        S2
*****
*YEAR          **
*ACCOUNT       **
*AMOUNT        **
*              **
*              **
*****
*****
```

## The REGION Data Source

The REGION data source lists account information for the east and west 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.

### The REGION Master File

```
FILENAME=REGION, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S1,$
FIELDNAME=ACCOUNT, , FORMAT=A4, $
FIELDNAME=E_ACTUAL, , FORMAT=15C,$
FIELDNAME=E_BUDGET, , FORMAT=15C,$
FIELDNAME=W_ACTUAL, , FORMAT=15C,$
FIELDNAME=W_BUDGET, , FORMAT=15C,$
```

### The REGION Structure Diagram

```
SECTION 01      STRUCTURE OF FOCUS  FILE REGION  ON 01/05/96 AT 15.18.48

          TOP
01      S1
*****
*ACCOUNT      **
*E_ACTUAL     **
*E_BUDGET     **
*W_ACTUAL     **
*             **
*****
*****
```

## The COURSES Data Source

The COURSES data source describes education courses. It consists of one segment, CRSESEG1. The field DESCRIPTION has a format of TEXT (TX).

### The COURSES Master File

```
FILENAME=COURSES, SUFFIX=FOC,           $
SEGNAME=CRSESEG1, SEGTYPE=S1,          $$
FIELDNAME=COURSE_CODE, ALIAS=CC,       FORMAT=A6,  FIELDTYPE=I,  $
FIELDNAME=COURSE_NAME, ALIAS=CN,       FORMAT=A30,  $
FIELDNAME=DURATION, ALIAS=DAYS,        FORMAT=I3,   $
FIELDNAME=DESCRIPTION, ALIAS=CDESC,    FORMAT=TX50,  $
```

### The COURSES Structure Diagram

```
SECTION 01
STRUCTURE OF FOCUS FILE COURSES ON 01/05/94 AT 15.20.59

      CRSESEG1
01      S1
*****
*COURSE_CODE *I
*COURSE_NAME **
*DURATION **
*DESCRIPTION **I
* **
*****
*****
```

## The EMPDATA Data Source

The EMPDATA data source contains organizational data about a company's employees. It consists of one segment, EMPDATA. Note the following:

- The PIN field is indexed.
- The AREA field is a temporary one.

## The EMPDATA Master File

```

FILENAME=EMPDATA, SUFFIX=FOC
SEGNAME=EMPDATA, SEGTYPE=S1
FIELDNAME=PIN, ALIAS=ID, FORMAT=A9, INDEX=I, $
FIELDNAME=LASTNAME, ALIAS=LN, FORMAT=A15, $
FIELDNAME=FIRSTNAME, ALIAS=FN, FORMAT=A10, $
FIELDNAME=MIDINITIAL, ALIAS=MI, FORMAT=A1, $
FIELDNAME=DIU, ALIAS=CDIU, 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.ZM, $
FIELDNAME=HIREDATE, ALIAS=HDAT, FORMAT=YMD, $
$
DEFINE AREA/A13=DECODE DIU (NE 'NORTH EASTERN' SE 'SOUTH EASTERN'
CE 'CENTRAL' WE 'WESTERN' CORP 'CORPORATE' ELSE 'INVALID AREA');$

```

## The EMPDATA Structure Diagram

```

SECTION 01 STRUCTURE OF FOCUS FILE EMPDATA ON 01/05/96 AT 14.49.09

      EMPDATA
01    S1
*****
*PIN          **I
*LASTNAME     **
*FIRSTNAME    **
*MIDINITIAL   **
*             **
*****
*****

```

## The EXPERSON Data Source

The EXPERSON data source contains personal data about individual employees. It consists of one segment, ONESEG.

### The EXPERSON Master File

```

FILE=EXPERSON      ,SUFFIX=FOC
SEGMENT=ONESEG, $
FIELDNAME=SOC_SEC_NO  ,ALIAS=SSN      ,USAGE=A9      ,,$
FIELDNAME=FIRST_NAME  ,ALIAS=FN      ,USAGE=A9      ,,$
FIELDNAME=LAST_NAME   ,ALIAS=LN      ,USAGE=A10     ,,$
FIELDNAME=AGE         ,ALIAS=YEARS   ,USAGE=I2      ,,$
FIELDNAME=SEX         ,ALIAS=        ,USAGE=A1      ,,$
FIELDNAME=MARITAL_STAT,ALIAS=MS      ,USAGE=A1      ,,$
FIELDNAME=NO_DEP     ,ALIAS=NDP     ,USAGE=I3      ,,$
FIELDNAME=DEGREE     ,ALIAS=        ,USAGE=A3      ,,$
FIELDNAME=NO_CARS    ,ALIAS=CARS    ,USAGE=I3      ,,$
FIELDNAME=ADDRESS    ,ALIAS=        ,USAGE=A14     ,,$
FIELDNAME=CITY       ,ALIAS=        ,USAGE=A10     ,,$
FIELDNAME=WAGE       ,ALIAS=PAY     ,USAGE=D10.2SM ,,$
FIELDNAME=CATEGORY   ,ALIAS=STATUS  ,USAGE=A1      ,,$
FIELDNAME=SKILL_CODE ,ALIAS=SKILLS  ,USAGE=A5      ,,$
FIELDNAME=DEPT_CODE  ,ALIAS=WHERE   ,USAGE=A4      ,,$
FIELDNAME=TEL_EXT    ,ALIAS=EXT     ,USAGE=I4      ,,$
FIELDNAME=DATE_EMP   ,ALIAS=BASE_DATE,USAGE=I6YMTD  ,,$
FIELDNAME=MULTIPLIER ,ALIAS=RATIO   ,USAGE=D5.3    ,,$
    
```

### The EXPERSON Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS   FILE EXPERSON ON 01/05/96 AT 14.50.58

          ONESEG
          S1
          *****
          *SOC_SEC_NO **
          *FIRST_NAME **
          *LAST_NAME  **
          *AGE        **
          *           **
          *****
          *****
    
```

## The TRAINING Data Source

The TRAINING data source contains training course data for employees. It consists of one segment, TRAINING. Note the following:

- The PIN field is indexed.
- The EXPENSES, GRADE, and LOCATION fields have the MISSING=ON attribute.

## The TRAINING Master File

```

FILENAME=TRAINING, SUFFIX=FOC
SEGNAME=TRAINING, SEGTYPE=SH3
FIELDNAME=PIN, ALIAS=ID, FORMAT=A9, INDEX=1, $
FIELDNAME=COURSESTART, ALIAS=CSTART, FORMAT=YMD, $
FIELDNAME=COURSECODE, ALIAS=CCOD, FORMAT=A7, $
FIELDNAME=EXPENSES, ALIAS=COST, FORMAT=D8.2, MISSING=ON,$
FIELDNAME=GRADE, ALIAS=GRA, FORMAT=A2, MISSING=ON,$
FIELDNAME=LOCATION, ALIAS=LOC, FORMAT=A6, MISSING=ON,$

```

## The TRAINING Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS   FILE TRAINING ON 12/12/94 AT 14.51.28

          TRAINING
01      SH3
*****
*PIN          **I
*COURSESTART **
*COURSECODE  **
*EXPENSES    **
*            **
*****
*****

```

## The PAYHIST File

The PAYHIST 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.

## The PAYHIST Master File

```
FILENAME=PAYHIST, SUFFIX=FIX
SEGMENT=PAYSEG,$
  FIELDNAME=SOC_SEC_NO, ALIAS=SSN, USAGE=A9, ACTUAL=A9,$
  FIELDNAME=DATE_OF_IN, ALIAS=INCDATE, USAGE=16YMTD, ACTUAL=A6,$
  FIELDNAME=AMT_OF_INC, ALIAS=RAISE, USAGE=D6.2, ACTUAL=A10,$
  FIELDNAME=PCT_INC, ALIAS=, USAGE=D6.2, ACTUAL=A6,$
  FIELDNAME=NEW_SAL, ALIAS=CURR_SAL, USAGE=D10.2, ACTUAL=A11,$
  FIELDNAME=FILL, ALIAS=, USAGE=A38, ACTUAL=A38,$
```

## The PAYHIST Structure Diagram

```
SECTION 01          STRUCTURE OF FIX   FILE PAYHIST  ON 01/05/96 AT 14.51.59

          PAYSEG
01      S1
*****
*SOC_SEC_NO **
*DATE_OF_IN **
*AMT_OF_INC **
 *PCT_INC   **
 *          **
*****
*****
```



## The COMASTER File

The COMASTER file is used to display the file structure and contents of each segment in a data source. Since COMASTER is used for debugging other Master Files, a corresponding FOCEXEC does not exist for the COMASTER file. Its segments are:

- FILEID lists file information.
- RECID lists segment information.
- FIELDID lists field information.
- DEFREC lists a description record.
- PASSREC lists read/write access.
- CRSEG lists cross-reference information for segments.
- ACCSEG lists DBA information.

## The COMASTER Master File

```

FILE=COMASTER, SUFFIX=COM,

SEGNAME=FILEID
FIELDNAME=FILENAME ,FILE ,A8 , ,,$
FIELDNAME=FILE SUFFIX ,SUFFIX ,A8 , ,,$

SEGNAME=RECID
FIELDNAME=SEGNAME ,SEGMENT ,A8 , ,,$
FIELDNAME=SEGTYPE ,SEGTYPE ,A4 , ,,$
FIELDNAME=SEGSIZE ,SEGSIZE ,I4 , A4,$
FIELDNAME=PARENT ,PARENT ,A8 , ,,$
FIELDNAME=CRKEY ,UREY ,A66, ,,$

SEGNAME=FIELDID
FIELDNAME=FIELDNAME ,FIELD ,A66, ,,$
FIELDNAME=ALIAS ,SYNONYM ,A66, ,,$
FIELDNAME=FORMAT ,USAGE ,A8 , ,,$
FIELDNAME=ACTUAL ,ACTUAL ,A8 , ,,$
FIELDNAME=AUTHORITY ,AUTHCODE ,A8 , ,,$
FIELDNAME=FIELDTYPE ,INDEX ,A8 , ,,$
FIELDNAME=TITLE ,TITLE ,A64, ,,$
FIELDNAME=HELPMESSAGE ,MESSAGE ,A256, ,,$
FIELDNAME=MISSING ,MISSING ,A4, ,,$
FIELDNAME=ACCEPTS ,ACCEPTABLE ,A255, ,,$
FIELDNAME=RESERVED ,RESERVED ,A44, ,,$

SEGNAME=DEFREC
FIELDNAME=DEFINITION ,DESCRIPTION ,A44, ,,$

SEGNAME=PASSREC, PARENT=FILEID
FIELDNAME=READ/WRITE ,RW ,A32, ,,$

SEGNAME=CRSEG, PARENT=RECID
FIELDNAME=CRFILENAME ,CRFILE ,A8 , ,,$
FIELDNAME=CRSEGNAME ,CRSEGMENT ,A8 , ,,$
FIELDNAME=ENCRYPT ,ENCRYPT ,A4 , ,,$

SEGNAME=ACCSEG, PARENT=DEFREC
FIELDNAME=DBA ,DBA ,A8 , ,,$
FIELDNAME=DBAFILE , ,A8 , ,,$
FIELDNAME=USER ,PASS ,A8 , ,,$
FIELDNAME=ACCESS ,ACCESS ,A8 , ,,$
FIELDNAME=RESTRICT ,RESTRICT ,A8 , ,,$
FIELDNAME=NAME ,NAME ,A66, ,,$
FIELDNAME=VALUE ,VALUE ,A80, ,,$

```

# The COMASTER Structure Diagram

SECTION 01

STRUCTURE OF EXTERNAL FILE COMASTER ON 12/12/94 AT 14.53.38

```

      FILEID
01      S1
*****
*FILENAME **
*FILE SUFFIX **
* **
* **
*****
      I
      +-----+
      I          I
      I RECID    I PASSREC
02  I N          07  I N
*****          *****
*SEGNAME **    *READ/WRITE **
*SEGTYPE **    * **
*SEGSIZE **    * **
*PARENT **     * **
* **          * **
*****          *****
      I
      +-----+
      I          I
      I FIELDID  I CRSEG
03  I N          06  I N
*****          *****
*FIELDNAME **  *CRFILENAME **
*ALIAS **      *CRSEGNAME **
*FORMAT **     *ENCRYPT **
*ACTUAL **     * **
* **          * **
*****          *****
      I
      I
      I
      I DEFREC
04  I N
*****
*DEFINITION **
* **
* **
* **
*****
      I
      I
      I
      I ACCSEG
05  I N
*****
*DBA **
*DBAFILE **
*USER **
*ACCESS **
* **
*****
*****

```

## The VideoTrk and MOVIES Data Sources

The VideoTrk data source tracks customer, rental, and purchase information for a video rental business. It can be joined to the MOVIES 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, $
```

### 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, $
```

# VideoTrk Structure Diagram

```

SECTION 01
                STRUCTURE OF FOCUS      FILE VIDEOTRK ON 05/21/99 AT 12.25.19

                CUST
01             S1
*****
*CUSTID       **
*LASTNAME     **
*FIRSTNAME    **
*EXPDATE      **
*             **
*****
                I
                I
                I
                I  TRANSDAT
02             I SH1
*****
*TRANSDATE    **
*             **
*             **
*             **
*             **
*****
                I
                +-----+
                I             I
                I SALES       I RENTALS
03             I S2           04   I S2
*****
*PRODCODE     **   *MOVIECODE  **I
*TRANSCODE    **   *COPY       **
*QUANTITY     **   *RETURNDATE **
*TRANSTOT     **   *FEE        **
*             **   *             **
*****
                *****
                *****

```

## MOVIES Structure Diagram

```
SECTION 01
                STRUCTURE OF FOCUS   FILE MOVIES   ON 05/21/99 AT 12.26.05

                MOVINFO
01             S1
*****
*MOVIECODE    **I
*TITLE        **
*CATEGORY     **
*DIRECTOR     **
*             **
*****
*****
```

## The VIDEOTR2 Data Source

The VIDEOTR2 data source tracks customer, rental, and purchase information for a video rental business. It is similar to VideoTrk but is a partitioned data source with both a Master and Access File and with a date-time field.

## The VIDEOTR2 Master File

```
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,      $
  FIELDNAME=EMAIL,       ALIAS=EMAIL,      FORMAT=A18,     $
SEGNAME=TRANSDAT,  SEGTYPE=SH1,  PARENT=CUST
  FIELDNAME=TRANSDATE,   ALIAS=OUTDATE,    FORMAT=HYMYDI,  $
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 VIDEOTR2 Access File

On CMS,

```
MASTER 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;
```

On MVS, the data set names include your user ID as the high-level qualifier:

```
MASTER VIDEOTR2
  DATANAME userid.VIDPART1.FOCUS
  WHERE DATE EQ 1991;

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

  DATANAME userid.VIDPART2.FOCUS
  WHERE DATE FROM 1999 TO 2000;
```

## The VIDEOTR2 Structure Diagram

STRUCTURE OF FOCUS

FILE VIDEOTR2 ON 09/27/00 AT 16.45.48

```

          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
*****          *****
*TRANSCODE   **  *MOVIECODE  ** I
*QUANTITY    **  *COPY        **
*TRANSTOT    **  *RETURNDATE **
*            **  *FEE         **
*            **  *            **
*****          *****
*            **  *            **

```



# The Gotham Grinds Data Sources

Gotham Grinds is a group of data sources that contain information about a specialty items company.

## The GGDEMOG Data Source

The GGDEMOG data source 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.

### The 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=AUGHHS298, 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=AUGHHI98, 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=P15T01998, ALIAS=E09, FORMAT=I09, TITLE='15 to 19',
DESC='Population 15 to 19 years old', $
FIELD=P20T02998, ALIAS=E10, FORMAT=I09, TITLE='20 to 29',
DESC='Population 20 to 29 years old', $
FIELD=P30T04998, ALIAS=E11, FORMAT=I09, TITLE='30 to 49',
DESC='Population 30 to 49 years old', $
FIELD=P50T06498, 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', $
```

## The GGDEMOG Structure Diagram

```
NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=    1 ( REAL=    1 VIRTUAL=    0 )
NUMBER OF FIELDS=     12 INDEXES=    1 FILES=      1
TOTAL LENGTH OF ALL FIELDS= 101
SECTION 01
                STRUCTURE OF FOCUS   FILE GGDEMOG   ON 09/17/96 AT 12.18.05

                GGDEMOG
01             S1
*****
*ST             ** I
*HH             **
*AVGHHSZ98     **
*MEDHHI98     **
*              **
*****
*****
```

## The GGORDER Data Source

The GGORDER data source contains order information for Gotham Grinds. It consists of two segments, ORDER01 and ORDER02, respectively.

### The 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 , $
```

## The GGORDER Structure Diagram

```

NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=  2 ( REAL=  1 VIRTUAL=  1 )
NUMBER OF FIELDS=   12 INDEXES=  0 FILES=   2
TOTAL LENGTH OF ALL FIELDS=  92
SECTION 01
      STRUCTURE OF FOCUS    FILE GGORDER  ON 09/17/96 AT 12.29.24

      GGORDER
01      S1
*****
*ORDER_NUMBER**
*ORDER_DATE **
*STORE_CODE **
*PRODUCT_CODE**
*           **
*****
      I
      I
      I
      I ORDER02
02      I KU
-----
:PRODUCT_ID :K
:PRODUCT_DES>:
:VENDOR_CODE :
:VENDOR_NAME :
:           :
:           :
:           :
-----

```

## The GGPRODS Data Source

The GGPRODS data source contains product information for Gotham Grinds. It consists of one segment, PRODS01.

### The 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=UCD, 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', $

```

## The GGPRODS Structure Diagram

```

NUMBER OF ERRORS=          0
NUMBER OF SEGMENTS=       1 ( REAL=    1 VIRTUAL=  0 )
NUMBER OF FIELDS=         7 INDEXES=   2 FILES=    1
TOTAL LENGTH OF ALL FIELDS=  63
SECTION 01
                STRUCTURE OF FOCUS   FILE GGPRODS   ON 09/17/96 AT 12.21.12

                GGPRODS
01             S1
*****
*PRODUCT_ID   **I
*VENDOR_CODE  **I
*PRODUCT_DES>***
*VENDOR_NAME  **
*              **
*****

```

## The GGSales Data Source

The GGSales data source contains sales information for Gotham Grinds. It consists of one segment, SALES01.

### The 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', $

```

## The GGSALES Structure Diagram

```

NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=   1 ( REAL=    1 VIRTUAL=    0 )
NUMBER OF FIELDS=    13 INDEXES=    5 FILES=    1
TOTAL LENGTH OF ALL FIELDS= 114
SECTION 01

```

STRUCTURE OF FOCUS FILE GGSALES ON 09/17/96 AT 12.22.19

```

          GGSALES
01      S1
*****
*SEQ_NO      **
*CATEGORY    **I
*PCD         **I
*REGION      **I
*           **
*****
*****

```

## The GGSTORES Data Source

The GGSTORES data source contains information for each of Gotham Grinds' 12 stores in the United States. It consists of one segment, STORES01.

### The 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', $

```

## The GGSTORES Structure Diagram

```
NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=   1 ( REAL=   1 VIRTUAL=   0 )
NUMBER OF FIELDS=    7  INDEXES=   2  FILES=    1
TOTAL LENGTH OF ALL FIELDS= 108
```

SECTION 01

STRUCTURE OF FOCUS FILE GGSTORES ON 09/17/96 AT 12.23.09

```
          GGSTORES
01      S1
*****
*STORE_CODE  **I
*STATE_      **I
*STORE_NAME  **
*ADDRESS1    **
*            **
*****
*****
```

---

## APPENDIX B

# Error Messages

### Topics:

- Accessing Error Files
- Displaying Messages Online

If you need to see the text or explanation for any error message, you can display it online in your FOCUS session or find it in a standard FOCUS ERRORS file. All of the FOCUS error messages are stored in eight system ERRORS files.

## 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 MVS, these files are the following members in the ERRORS PDS:

- FOT004
- FOG004
- FOM004
- FOS004
- FOA004
- FSQLXLT
- FOCSTY
- FOB004



## Displaying Messages Online

To display a message online, issue the following query command at the FOCUS command level

? *n*

where *n* is the message number.

The message number and text will display along with a detailed explanation of the message (if one exists). For example, issuing the following command

? 210

displays the following:

**(FOC210) THE DATA VALUE HAS A FORMAT ERROR:**

An alphabetic character has been found where all numerical digits are required.

---

## APPENDIX C

# User Exits for a Non-FOCUS Data Source

### Topics:

- The Dynamic and Re-Entrant Private User Exit of the FOCUSAM Interface
- User-coded Data Access Modules
- Re-Entrant VSAM Compression Exit: ZCOMP1

This appendix describes three ways to read non-FOCUS data sources with user-written procedures.

## The Dynamic and Re-Entrant Private User Exit of the FOCSAM Interface

The FOCSAM Interface contains a user exit that can be invoked as an alternative to the lowest-level retrieval routines that are part of the interface. This interface is responsible for data access to VSAM and QSAM structures. The Master File would specify SUFFIX=VSAM or SUFFIX=FIX. This exit makes it possible to combine user-written code, devoid of any dependence on internal FOCUS structures, with the logical retrieval functions of the FOCSAM Interface, such as record selection logic, treatment of missing records in multi-record files, JOINS between various types of files, and so forth. The major features of the GETPRV exit are as follows:

1. Through the CONTEXT parameter, the exit supports reentrancy, providing the benefit of reduced storage requirements as well as enhanced invocation performance.
2. Support for multiple concurrent exit processors is provided through an Access File where a user exit module can be named on a per Master File basis.
3. The user exit is dynamically called at execution time, avoiding the need to modify FOCUS after each upgrade or application of maintenance. There is no need for link-edits to FOCUS.
4. An initialization call allows the exit code to perform initial housekeeping.
5. 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.
6. 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 be able to do the following:

1. Obtain a record, given a full value of the key. The key is presumed to be unique (direct read).
2. Obtain a record that is greater than or equal to a given value of the full or partial key (generic read).
3. Obtain the next logical record, starting from the current position in the file (sequential read). Successive sequential reads must return records that are in ascending sequence (bit by bit) on the key field.
4. Direct and generic reads that retrieve records must establish the starting position in the file for subsequent sequential reads. Direct reads that fail to retrieve the requested records need not establish these positions.
5. For the open file request (O), it must logically or physically open the file.
6. If the logical end-of-file is reached as a result of a sequential read, an end-of-file signal must be returned. Subsequent sequential reads must return end-of-file signals rather than error indications, for example, when processing a JOIN.
7. A 'close' function must be provided that results in the release of all resources and loss of position in the file, so that subsequent open requests succeed.
8. Successive close calls must be innocuous. Be prepared to close a file that is already closed.
9. A unique area must be obtained, for example, GETMAIN, and maintained using the CONTEXT parameter on a per DDNAME basis.
10. The code must be serially reusable and reentrant. It must be linked AMODE 31.

## Implementation

The Dynamic GETPRV exit is linked as a separate module and loaded or called from FOCUS.

## The Master File

The Master File for data to be accessed using this user exit is exactly the same as the description of any other data source read by the FOCSAM Interface except that the SUFFIX must specify PRIVATE. All other READ ONLY features of the FOCSAM Interface are fully supported.

### Example

#### Sample Master File

```
FILE=filename, SUFFIX=PRIVATE,$
SEGNAME=ROOT , SEGTYPE=S0,$
GROUP=keyname , ALIAS=KEY , USAGE=xx, ACTUAL=xx ,$
FIELD=fieldname1, ALIAS=aliasname1, USAGE=xx, ACTUAL=xx ,$
FIELD=fieldname2, ALIAS=aliasname2, USAGE=xx, ACTUAL=xx ,$
```

**Note:** SUFFIX=PRIVATE. No other options will invoke the exit.

## The Access File

An Access File is required and provides a pointer to the actual name of the private exit. The PDS used for the Access File must be allocated to DDNAME ACCESS.

### Example

#### Sample Access File

```
MODNAME=pgmname,$
```

### Example

#### Allocating an Access File

```
//ACCESS DD DSN=access.file.pds.name,DISP=SHR
```

## Calling Sequence

The user-coded retrieval routine is written as a standalone program. There are no limitations to program name other than standard IBM rules. We recommend that the program be written in a language that fully supports reentrancy, such as ASSEMBLER or C. The parameter list is as follows:

### NCHAR

Full-word binary integer, posted by the exit. A positive number indicates the length in bytes of the record obtained; zero indicates no record or end-of-file (there is no difference). Must be non-zero for every successful read. Used by read options (S), (E), and (G). Do not modify the pointer. Instead, modify the location addressed by the pointer.

### DDNAME

8-byte character argument, posted by FOCSAM, corresponding to the FOCUS 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. Do not modify the pointer. Instead, modify the location addressed by the pointer.

### KEY

Full-word binary integer posted by FOCSAM, containing the full value of the key for direct or generic reads. Not significant for sequential reads. The key value is left justified and less than 255 bytes long.

Note that the exit must know the true key length and its format. Used with options (E) and (G). Do not modify this parameter.

**OPT**

Four-byte character argument, posted by FOCSAM, indicating the type of call. Do not modify this parameter. The OPTions are as follows:

**READ OPTIONS**

'S ' =sequential read.

'E ' =direct read (EQ).

'G ' =generic read (GE).

**CONTROL OPTIONS**

'O ' =OPEN file.

'R ' =OPEN request (position) used for recursive JOINS.

'C ' =CLOSE file.

'F ' =FIN for FOCUS -- final housekeeping should be done here.

These control and read arguments must include three trailing blanks.

**CONTEXT**

Full-word binary integer, posted by FOCSAM, which points to a work area described below. Do not modify this parameter.

**Note:** The parameters passed to the exit are not delimited with the final parameter having the high-order bit set on. Make sure that your program does not scan for this high-order bit.

## Work Area Control Block

**EYECATCH**

An 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).

**PFRL**

Full-word binary integer, posted by the exit. Generally set during OPTion (R) processing by the exit program and returned unchanged by subsequent FOCSAM calls for OPTions (E), (G), and (S) and is unique for each view within the above PFACB parameter. This is generally used for logical file context.

This parameter is not valid for OPTion (F).

**KEYLENF**

Full-word binary integer, posted by the exit. Generally set during OPTion (O) processing by the exit program and should contain the whole key length for the file.

**KEYLENR**

Full-word binary integer, posted by the exit. Generally set during OPTions (G) and (E) processing by the exit program and should contain the actual key length for a direct read.

**ERRTEXT**

Full-word binary integer, posted by the exit. Should contain the absolute starting address of a message. FOCUS displays this message if the RC parameter returned by the exit is non-zero and for OPTions (E), (G), and (S).

**ERRTEXTL**

Full-word binary integer, posted by the exit. It should contain the length of the above ERRTEXT message.

**INDEX**

Full-word binary integer, posted by FOCSAM.

This option contains the index # by which to access the file.

- 0 Primary key in Master File in Master -- KEY-DKEY
- 1,2,e Secondary indexes in the Master File
- tc KEY1-DKEY1 or KEY2-DKEY2, and so on, and INDEX=I

**RESERVE1**

Full-word binary integer, reserved for future use.

**USERID**

Full-word binary integer, posted by FOCSAM. This userid will only be present if found in the central site security environment.

**RESERVE2**

Full-word binary integer, reserved for future use.



**Example**            **Sample Assembler DSECT**

```

GETPRVPL DS      0F
NCHAR@   DS      A
DDN@     DS      A
ABUF@    DS      A
RC@      DS      A
KEY@     DS      A
OPT@     DS      A
CONTEXT@ DS      A
          TITLE 'GETPRV CONTEXT'
CONTEXTD DSECT
EYE      DS      CL8   eye catcher literal
PFMCB@   DS      A     handle
PFACB@   DS      A     file open handle
PFRPL@   DS      A     file retrieval handle
KEYLEN_F DS      F     key length for file
KEYLEN_R DS      F     key length for this request
RETTEXT@ DS      A     pointer to returned message
LRETTEXT DS      F     length of returned message
INDEX    DS      F     index for file access - 0 primary 1... secondary
          DS      A     reserved
USERID   DS      CL8
          DS      2F    reserved
          SPACE 1
    
```

**Example**            **Sample C Declaration Required For Invoking FFUN Function**

```

/*
control block for additional info
*/
typedef struct getprv_inf_s {
char      eye[8];      /* I: eye catcher "PRIVATE " */
Pvoid     pfmcb;       /* o: p' to handle forgetprv  */
/* set up by user at first option O */
/* I: p' to handle for getprv  */
/* passed to user by all other calls*/
Pvoid     pfacb;       /* o: p' to handle for file  */
/* set up by user at option O      */
/* i: p' to handle for file */
/* passed to user at option C,R,E,G,S */
Pvoid     pfrpl;       /* o: p' to handle for request */
/* set up by user at option R      */
/* i: p' to handle for request */
/* passed to user at option E,G,S */
long      keylen_fil; /* I: key length (whole) for the file . */
/* used at option O . */
long      keylen_req; /* I: key length for the direct read request*/
/* used at direct read options G,E */
char      *rettext;   /* O: a'native db error msg text */
long      lrettext;   /* O: l'native db error msg text */
    
```

```
long      index;      /* I:index # by which to access file :
                      = 0      - primary key
                      in master - KEY|DKEY
                      = 1,2,... - secondary indexes
                      in master - KEY1|DKEY1 or KEY2|DKEY2 ...
                      and INDEX=I */

long      res1[1];    /* reserved */
char      userid[8]; /* user id */
long      res2[2];    /* reserved */
} getprv_inf_t;
/*

*/
typedef void FFUN getprv_t (
    long      *nchar      /* out: length of data record read. =0 */
                      /* if eof or no rec found */
                      /* used in all read options S,E,G */
, char      *ddn         /* in : ddname to read */
                      /* used in all options except F */
, char      **abuf       /* out: a' buffer */
                      /* used in all read options S,E,G */
, long      *rc          /* out" return code. = 0 if ok */
                      /* used in all options */
, char      *key         /* in: key value for read */
                      /* used in read options E,G */
, char      *opt         /* in: read option : */
                      /* S sequential read */
                      /* G GE read */
                      /* E EQ read */
                      /* control options */
                      /* O open file */
                      /* R open request (position)*/
                      /* C close file */
                      /* F fin of focus */
, getprv_inf_t *        /* in/out other info .see above */
);
#endif
```

## User-coded Data Access Modules

A user routine may be written to provide records to the FOCUS report writer from any non-standard data source that cannot be directly described to FOCUS using the Master File. The record, which can come from any data source, is treated by FOCUS exactly as if it had come from a FOCUS 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**

(4 byte integer) If  $LCHAR > 0$ , LCHAR is the number of characters in the record passed back to FOCUS. If  $LCHAR = 0$ , the user routine is telling FOCUS that an end-of-file has been encountered and the retrieval is complete. If  $LCHAR = -N$ , the user routine is telling FOCUS that the buffer contains an error message of length N, to be printed out, and that the user routine should not be called again.  $LCHAR = -1$  is reserved for Information Builders' use.

**BUF**

This parameter is a 4096 byte buffer provided by FOCUS to receive the record from the user routine.

**OFFSET**

(4 byte integer) If the user routine puts data in BUF, OFFSET should be set to 0 each time the user routine is called. If the user routine provides its own buffer or buffers, OFFSET is the address of the user's buffer minus the address of BUF. A utility called IADDR is provided to compute an address. In FORTRAN, for example, one could code:

```
OFFSET = IADDR (MYBUF) - IADDR (BUF)
```

**RECTYP**

(4 byte integer) The number of the FOCUS segment corresponding to the record being presented to FOCUS, set by the routine. These numbers correspond to either the list obtained by issuing '? FILE filename' or the field SEGNO resulting from a CHECK FILE filename HOLD.

**NERRX**

(4 byte integer) Flag set by FOCUS. If NERRX < 0, FOCUS is directing the user routine to shut down (for example, close all files) and not provide any more records. On return, FOCUS will not call the subroutine again.

**CSEGX REGI NFIND MATCH**

Reserved for Information Builders' use.

**IGNOR**

(4 byte integer) FOCUS will reject any segment whose number (see RECTYP) is greater than or equal to IGNOR. IGNOR is set by FOCUS, based on the segments referenced in the request, but may be reset by the user routine.

**NUMFLD NUMLEV**

Reserved for Information Builders. Use CVT.

On CMS, FOCUS will look for MYREAD TEXT on any accessed CMS minidisk. If MYREAD TEXT is not found, FOCUS will then look for a member MYREAD within all TXTLIB files currently pointed at by a GLOBAL TXTLIB command.

On TSO, the MYREAD object code, and any other routine that it calls, must be link-edited into a load module whose member name in the load library is MYREAD. This load library can be allocated as ddname USERLIB or concatenated with the STEPLIB program library.

## Re-Entrant VSAM Compression Exit: ZCOMP1

This re-entrant ZCOMP1 exit works with compressed VSAM and sequential files. It requires an initial call to ZCOMP0 for initial housekeeping and a USERWD parameter to anchor storage.

The ZCOMP1 user exit was designed to provide an exit that can be user-supplied in order to decrypt coded fields, expand compressed records, and accomplish any other type of user-specific data manipulation required. This exit is designed to access all files readable by the FOCSAM Interface. It is the user's responsibility to write and maintain this exit.

There are no special Master File requirements. SUFFIX can equal VSAM (for KSDS or ESDS files), FIX (for sequential files), or PRIVATE (for file access through the GETPRV user exit).

### Linking ZCOMP1

After you write a ZCOMP1 user exit, it must be linked with VVSET. This process of linking ZCOMP1 into VVSET can be accomplished by using the GENFSAM EXEC (for VM), or the GENFSAM JCL (for MVS) that is found in the FOCCTL.DATA data set. Note that GENFSAM is designed to link in both ZCOMP1 and GETPRV user exits at the same time. Should you only be implementing one of them, the VM EXEC generates the correct linkage to the routine required, whereas in the MVS JCL, the appropriate lines in the GENFSAM member must be commented out (these would be the INCLUDE OBJECT statements). ZCOMP1 can be linked re-entrant if you plan to use the USERWD parameter.

### What Happens When You Use ZCOMP1

The FOCSAM Interface reads the record for the allocated data source. Upon a successful read, FOCSAM calls ZCOMP0, if it exists, with the parameters listed below so that initial housekeeping can be performed. All subsequent calls are to ZCOMP1 with the same parameter list.

The ZCOMP1 exit is responsible for determining what to do with the parameter information it receives. The DDNAME can be used to determine whether the associated data source needs to be decompressed or not. If not, the user exit typically moves A(IRECLEN) to A(ORECLEN) and A(A(IREC)) to A(A(OREC)) and returns to FOCSAM with a zero A(STATCODE). If decompression or any other processing is required, it is the responsibility of the user exit to do so.

After the user exit has completed its processing, it should return with either the A(ORECLEN), A(A(OREC)) and a zero status code or with a non-zero status code which gives the following message:

```
(FOC1150) ZCOMP DECOMPRESS ERROR: status
```

**Note:** This error terminates a TABLE request.

## ZCOMP1 Parameter List

| Parameter    | Description                            | Length           |
|--------------|----------------------------------------|------------------|
| A(STATCODE)* | Pointer to fullword binary status code | 4 byte integer   |
| A(DDNAME)    | Pointer to 8 byte file name in use     | 8 byte character |
| A(USERID)    | Reserved for future use                | 8 byte character |
| A(IRECLEN)   | Pointer to length of original record   | 4 byte integer   |
| A(A(IREC))   | Pointer to pointer to original record  | 4 byte integer   |
| A(ORECLEN)*  | Pointer to length of revised record    | 4 byte integer   |
| A(A(OREC))*  | Pointer to pointer to revised record   | 4 byte integer   |
| A(USERWD)**  | Pointer to fullword                    | 4 byte integer   |

\* The user supplies these parameters.

\*\* This parameter can be used to anchor user storage for re-entrant processing.

**Note:** Never modify the primary pointers, but rather the pointers or values they point to.

Note that upon entry to ZCOMP1, the ORECLen and OREC parameters are NULL. It is the responsibility of the user to fill these in correctly.

While processing the FOCUS FIN command, a call is placed to the ZCOMP2 entry point, which provides the user with the ability to do any other global cleanup required.

The parameters returned by ZCOMP1 are not validated. It is the responsibility of the user routine to ensure that valid addresses and lengths are returned to FOCUS from ZCOMP1.

Unpredictable results occur if incorrect parameters are passed back from the routine.

---

## APPENDIX D

# Rounding in FOCUS

### Topics:

- Data Storage and Display
- Rounding in Calculations and Conversions

This appendix describes how FOCUS numeric fields store and display data, how rounding occurs in calculations, and what happens in conversion from one format to another.

## Data Storage and Display

Values are rounded before storage or before display, depending on the format. Integer fields (format I) and packed decimal fields (format P) are rounded before they are stored. Floating-point fields (formats F and D) are stored as entered and rounded for display.

When a final decimal digit is less than 5, the data value rounds down. A data value with a final digit of 5 or greater rounds up. The following rounding algorithm is used:

1. The incoming value is multiplied by 10.
2. This multiplication repeats the same number of times as the number of decimal places in the target format. For example, if 123.78 is input to a packed decimal field with one decimal place, it is multiplied by 10 once:

1237.8

3. Next, 0.5 is added if the incoming value is positive or subtracted if the incoming value is negative:

1237.8 + 0.5 = 1238.3

or, if the input value was -123.78,

-1237.8 - 0.5 = -1238.3

4. The value is truncated, and the decimal is shifted to the left.

123.8

or, if the original value was negative,

-123.8

The following chart illustrates the rounding differences between FOCUS numeric field formats. Subsequent topics discuss these differences in detail.

| Format | Type                               | Format | Input  | Stored           | Display |
|--------|------------------------------------|--------|--------|------------------|---------|
| I      | Integer                            | I3     | 123.78 | 0124             | 124     |
| F      | Floating-Point<br>Single-Precision | F3     | 123.78 | 123.7800         | 124     |
|        |                                    | F5.1   | 123.78 | 123.7800         | 123.8   |
| D      | Floating-Point<br>Double-Precision | D3     | 123.78 | 123.780000000000 | 124     |
|        |                                    | D5.1   | 123.78 | 123.780000000000 | 123.8   |
| P      | Packed                             | P3     | 123.78 | 0000124          | 124     |
|        |                                    | P5.1   | 123.78 | 00001238         | 123.8   |



## Integer Fields: Format I

An integer value entered with no decimal places is stored as entered.

When a value with decimal places is entered into an integer field using a transaction, that value is rounded, and the result is stored. If the fractional portion of the value is less than 0.5, the value is rounded down; if the fractional portion of the value is greater than or equal to 0.5, the value is rounded up. For example, if a value entered in a CRTFORM is 123.78, the value stored is 124.

However, if an integer field is computed, the decimal portion of the resulting value is truncated, and the integer portion of the answer is stored (or printed). For example, if the result of a calculation is 123.78, the value stored is 123.

## Floating-Point Fields: Formats F and D

Format type F describes single-precision floating-point numbers stored internally in 4 bytes. Format F is comparable to COBOL COMP-1. Format type D describes double-precision floating-point numbers stored internally in 8 bytes. Format D is comparable to COBOL COMP-2.

Formats F and D store as many decimal places as are input, up to the limit of the field's storage. Floating-point fields are stored in a logarithmic format. The first byte stores the exponent; the remaining 3 or 7 bytes store the mantissa, or value.

When the number of decimal places input is greater than the number of decimal places specified in the format, F and D field values are stored as they are input, up to the limit of precision. These values are rounded for display according to the field format. For example, if 123.78 is entered in a floating-point field with one decimal place, 123.78 is stored, and 123.8 is displayed.

Format D is more accurate than format F for larger numbers, since D fields can store up to 15 significant digits, and F fields are not accurate beyond 8 digits.

For floating-point fields (format D or F), the stored values of decimal numbers are in hexadecimal and may convert to a value very slightly less than the actual decimal number. When the final digit is 5, these numbers may round down instead of up.

## Packed Decimal Format: Format P

In packed-decimal format (format type P), each byte contains two digits, except the last byte, which contains a digit and the sign (D for negative numbers, C for positive). Packed fields are comparable to COBOL COMP-3.

Packed field values are rounded to the number of digits specified in the field format before they are stored.

When the number of decimal places input is greater than the number that can be stored, P field values are rounded first, then stored or displayed.

Packed fields are precisely accurate when sufficient decimal places are available to store values. Otherwise, since values are rounded before being stored, accuracy cannot be improved by increasing the number of digits displayed. For example, if 123.78 is input to a packed field with 1 decimal place, 123.8 is stored. If the field's format is then changed to P6.2 using a COMPUTE or DEFINE, 123.80 will be displayed. If the field's format is changed to P6.2 in the Master File, 12.38 is displayed.

### *Example*

### Storage and Display

For floating-point fields (format D or F), the stored values of decimal numbers are in hexadecimal and may convert to a value very slightly less than the actual decimal number. When the final digit is 5, these numbers may round down instead of up.

The following example shows an input value with two decimal places, which is stored as a packed field with two decimal places, a packed field with one decimal place, and a D field with one decimal place:

#### Master File

```
FILE=FIVE, SUFFIX=FOC
SEGNAME=ONLY, SEGTYPE=S1,$
FIELD=PACK2,,P5.2,$
FIELD=PACK1,,P5.1,$
FIELD=DOUBLE1,,D5.1,$
```

**Program to Load Data**

This MODIFY creates a file with three fields: a P field with two decimal places, a P field with one decimal place, and a D field with one decimal place. The same data values are then loaded into each field.

```
CREATE FILE FIVE

MODIFY FILE FIVE
FIXFORM  PACK2/5 PACK1/5 DOUBLE1/5
MATCH PACK2
  ON MATCH REJECT
  ON NOMATCH INCLUDE
DATA
1.05 1.05 1.05
1.15 1.15 1.15
1.25 1.25 1.25
1.35 1.35 1.35
1.45 1.45 1.45
1.55 1.55 1.55
1.65 1.65 1.65
1.75 1.75 1.75
1.85 1.85 1.85
1.95 1.95 1.95
END
```

**TABLE Request**

This TABLE request prints the values and a total for all three fields.

```
TABLE FILE FIVE
PRINT PACK2 PACK1 DOUBLE1
ON TABLE SUMMARIZE
END
```

The following report results:

```
PAGE      1

PACK2  PACK1  DOUBLE1
-----  -----  -----
 1.05   1.1    1.0
 1.15   1.2    1.1
 1.25   1.3    1.3
 1.35   1.4    1.3
 1.45   1.5    1.4
 1.55   1.6    1.5
 1.65   1.7    1.6
 1.75   1.8    1.8
 1.85   1.9    1.8
 1.95   2.0    1.9

TOTAL
15.00  15.5    15.0
```

The PACK2 values are not rounded. They are stored and displayed as they were entered. For example, 1.15 is stored internally as:

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 00 | 00 | 00 | 00 | 00 | 00 | 11 | 5C |
|----|----|----|----|----|----|----|----|

The PACK1 values are rounded when stored and also when displayed. For example, the incoming value 1.15 is rounded to 1.2, and stored internally as:

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 00 | 00 | 00 | 00 | 00 | 00 | 01 | 2C |
|----|----|----|----|----|----|----|----|

All of the DOUBLE1 values, except 1.25 and 1.75, are stored as repeating decimals in hex. For example, 1.15 is stored internally as:

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 41 | 12 | 66 | 66 | 66 | 66 | 66 | 66 |
|----|----|----|----|----|----|----|----|

The 41 in the first byte is equivalent to 64 in hex plus the exponent. The last seven bytes are the mantissa as converted to hex by the S/390.

The DOUBLE1 values 1.25 and 1.75 are not repeating decimals internally. They are terminating decimals in hex, so they round up, when displayed, to 1.3 and 1.8 respectively. For example, 1.25 is stored internally as:

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 41 | 14 | 00 | 00 | 00 | 00 | 00 | 00 |
|----|----|----|----|----|----|----|----|

Since the PACK1 values are rounded up before they are stored, the PACK1 total is 0.5 higher than the PACK2 total. The D field total is the same as the PACK2 total because the D field values are stored as input, and then rounded for display.

## Rounding in Calculations and Conversions

All computations are done in floating-point arithmetic. Packed fields are converted to D internally, then back to P.

When a field with decimal places is computed to an integer field, the decimal places are truncated, and the resulting value is the integer part of the input value.

When a field with decimal places is computed from one format to another, two conversions take place:

1. First the field is converted internally to floating-point notation.
2. Second, the result of this conversion is converted to the specified format. At this point, the rounding algorithm described previously is applied.

**Example****Redefining Field Formats**

The following example illustrates some differences in the way packed fields, floating-point fields, and integer fields are stored and displayed. It also shows database values redefined to a format with a larger number of decimal places.

**Master File**

```
FILE=EXAMPLE, SUFFIX=FOC
SEGNAME=ONLY, SEGTYPE=S1,$
FIELD=PACKED2,,P9.2,$
FIELD=DOUBLE2,,D9.2,$
FIELD=FLOAT2,,F9.2,$
FIELD=INTEGER,,I9,$
```

**Program to Load Data**

This MODIFY creates a file with four fields: a P field with two decimal places, a D field with two decimal places, an F field with two decimal places, and an integer field. The same data values are then loaded into each field.

```
CREATE FILE EXAMPLE

MODIFY FILE EXAMPLE
FIXFORM PACKED2/9 X1 DOUBLE2/9 X1 FLOAT2/9 X1 INTEGER/9
MATCH PACKED
  ON MATCH REJECT
  ON NOMATCH INCLUDE
DATA
1.6666666 1.6666666 1.6666666 1.6666666
125.16666 125.16666 125.16666 125.16666
5432.6666 5432.6666 5432.6666 5432.6666
4.1666666 4.1666666 4.1666666 4.1666666
5.5      5.5      5.5      5.5
106.66666 106.66666 106.66666 106.66666
7.2222222 7.2222222 7.2222222 7.2222222
END
```

### Report Request

A DEFINE command creates temporary fields that are equal to PACKED2, DOUBLE2, and FLOAT2, with redefined formats containing four decimal places instead of two. These DEFINE fields illustrate the differences in the way packed fields and floating-point fields are stored and displayed.

The request prints the values and a total for all four database fields, and for the three DEFINE fields.

```

DEFINE FILE EXAMPLE
PACKED4/P9.4=PACKED2;
DOUBLE4/D9.4=DOUBLE2;
FLOAT4/D9.4=FLOAT2;
END

TABLE FILE EXAMPLE
PRINT PACKED2 PACKED4 DOUBLE2 DOUBLE4 FLOAT2 FLOAT4 INTEGER
ON TABLE SUMMARIZE
END
    
```

The resulting report follows:

PAGE 1

| PACKED2 | PACKED4   | DOUBLE2  | DOUBLE4    | FLOAT2  | FLOAT4     | INTEGER |
|---------|-----------|----------|------------|---------|------------|---------|
| -----   | -----     | -----    | -----      | -----   | -----      | -----   |
| 1.67    | 1.6700    | 1.67     | 1.6667     | 1.67    | 1.6667     | 2       |
| 125.17  | 125.1700  | 125.17   | 125.1667   | 125.17  | 125.1667   | 125     |
| 5432.67 | 5432.6700 | 5,432.67 | 5,432.6666 | 5432.66 | 5,432.6641 | 5433    |
| 4.17    | 4.1700    | 4.17     | 4.1667     | 4.17    | 4.1667     | 4       |
| 5.50    | 5.5000    | 5.50     | 5.5000     | 5.50    | 5.5000     | 6       |
| 106.67  | 106.6700  | 106.67   | 106.6667   | 106.67  | 106.6667   | 107     |
| 7.22    | 7.2200    | 7.22     | 7.2222     | 7.22    | 7.2222     | 7       |
| TOTAL   |           |          |            |         |            |         |
| 5683.07 | 5683.0700 | 5,683.06 | 5,683.0555 | 5683.04 | 5,683.0529 | 5684    |

In this example, the PACKED2 sum is an accurate sum of the displayed values, which are the same as the stored values. The PACKED4 values and total are the same as the PACKED2 values.

The DOUBLE2 sum looks off by .01; it is not the sum of the printed values but a rounded sum of the stored values. The DEFINED DOUBLE4 values show that the DOUBLE2 values are actually rounded from the stored values. The DOUBLE4 values and sum show more of the decimal places from which the DOUBLE2 values are rounded.

The FLOAT2 total seems off by .03. Like the DOUBLE2 total, the FLOAT2 total is a rounded total of the stored FLOAT2 values. F fields are not accurate beyond 8 digits, as the FLOAT4 column shows.

The integer sum is an accurate total. Like packed fields, the storage values and displayed values are the same.

## DEFINE and COMPUTE

DEFINE and COMPUTE may give different results for rounded fields. DEFINE fields are treated like database 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 database field values. The printed total, 1420.765, is the sum of the printed DEFP3 values, just as the PACKED2 total is the sum of the printed PACKED2 values.

The COMPP3 field is the result of a COMPUTE. The printed total, 1420.767, is calculated from the total sum of PACKED2 (5683.07 / 4).

# Index

## Symbols

- \$ VIRT attribute, 1-5 to 1-6
- \$BOTTOM keyword, 10-6 to 10-8
- &QUIT variable, 10-36
- ? USE command, 9-14

## A

- A data type, 4-20
- Absolute File Integrity, 10-40 to 10-41
  - implementing, 10-41
  - shadow paging, 10-40 to 10-41
- ACCBLN parameter, 4-46
- ACCEPT attribute, 4-46 to 4-47, 6-15
  - FOCUS data sources, 6-15
- ACCEPTBLANK parameter, 4-46
- ACCESS attribute, 10-12
- Access File attributes, 6-22
  - DATANAME, 6-22 to 6-25
  - LOCATION, 6-22 to 6-25
  - MASTERNAME, 6-22 to 6-25
  - WHERE, 6-22 to 6-25
- Access Files, 1-2, 3-8, 6-22
  - applications and, 1-3
  - creating, 1-4
  - for FOCSAM interface, C-4
  - identifying, 6-21
  - specifying segment names, 3-4
  - VIDEOTR2, A-27
- ACCESSFILE attribute, 2-1, 6-21
  - DATASET attribute and, 2-5
- accounting, 10-37
  - resource limitation, 10-37
  - UACCT exit routine, 10-40
- ACTUAL attribute, 4-41

- ACTUAL format, 4-42 to 4-43, 5-45
- AIX (alternate index names) for VSAM data sources, 5-44
- ALIAS attribute, 4-10 to 4-11
- aliases of fields, 4-10 to 4-11
- ALLOWCVTERR parameter, 4-31
- alphanumeric data type, 4-20 to 4-21
- alternate column titles, 4-49
- alternate file views, 3-31 to 3-32, 3-34
  - CHECK FILE command and, 8-2
- alternate index names (AIX) for VSAM data sources, 5-44
- alternate indexes for VSAM data sources, 5-42 to 5-44
- ancestral segments, 3-13
- applications and data descriptions, 1-2 to 1-3, 2-1
- AS phrase, 8-10
  - CHECK HOLD option, 8-10
- attributes, 10-2
  - database security, 10-2
- AUTODATE field, 6-5, 6-12 to 6-14
- AUTOPATH parameter, 6-3

## B

- base date, 4-30
- blank lines between declarations, 1-10
- blank spaces in declarations, 1-10
- BUFND parameter, 5-41
- BUFNI parameter, 5-41



**C**

calculations on dates, 4-29

CAR data source, A-11

CHECK FILE command, 8-2, 8-4  
DBA and, 10-13  
DEFINE fields and, 8-11  
external data sources and, 8-4  
HELPMESSAGE attribute and, 8-10  
HOLD ALL option, 8-8 to 8-10  
HOLD option, 8-8 to 8-10  
long field names and, 4-5  
non-FOCUS data sources and, 8-4  
PICTURE option, 8-5, 8-7  
TAG attribute and, 8-10  
TITLE attribute and, 8-10  
virtual fields and, 8-11

child segments, 3-8

CLUSTER component, 5-1

column title substitutions, 4-49

columns in relational tables, 1-2

COMASTER data source, A-21

COMBINE command, 10-25  
data security, 10-25 to 10-26, 10-28

comma-delimited data sources, 5-2, 5-4  
repeating fields, 5-9 to 5-10

commands, 7-17  
CHECK FILE, 8-2, 8-4  
JOIN, 7-17  
USE, 9-1 to 9-5  
user access levels, 10-13

comments in Master Files, 1-11

common errors in Master Files, 8-4

COMPUTE command, D-9  
rounding, D-9

concatenation, 9-9 to 9-11  
data sources, 9-9 to 9-11

converting date values, 4-29

COURSES data source, A-16

CREATE command, 10-15

creating data descriptions, 1-4

cross-referenced relationships, 3-9

cross-referenced segments, 7-10 to 7-11, 7-13  
joining to, 7-19, 7-22  
recursive join structures, 7-23

currency symbols, 4-17 to 4-18

**D**

D data type, 4-14

data access, 9-8  
levels of, 10-12, 10-17  
read-only, 9-8  
security attributes, 10-2  
via a user-coded routine, C-10

data buffers for VSAM data sources, 5-41

data descriptions, 1-1 to 1-2, 2-1  
applications and, 1-3  
creating, 1-4  
field declarations, 1-4, 4-2  
field relationships, 1-4  
file declarations, 1-3, 2-1  
Master Files, 1-3, 1-9

data display, D-2, D-4

data encryption, 10-31  
performance considerations, 10-31

data relationships, 3-13  
alternate views, 3-31 to 3-32, 3-34  
describing, 3-7  
logical, 3-7  
many-to-many, 3-22 to 3-23  
multiple paths, 3-14  
one-to-many, 3-19 to 3-20  
one-to-one, 3-16 to 3-17  
parent-child, 3-9 to 3-10  
physical, 3-7  
recursive, 3-27 to 3-28  
root, 3-12  
single paths, 3-13

- data security, 10-1, 10-3 to 10-4
  - access levels, 10-12, 10-17
  - central Master File, 10-25 to 10-26
  - CHECK FILE command and, 10-13
  - COMBINE command and, 10-25 to 10-26, 10-28
  - encrypting Master Files, 10-30
  - encrypting segments, 10-31
  - filters, 10-28
  - JOIN command and, 10-25 to 10-26, 10-28
  - passwords, 10-10
  - RESTRICT command, 10-32
  - restricting access, 10-18 to 10-24
  - special considerations, 10-4
- data source declarations, 2-1
- data source security, 9-8
  - read-only access, 9-8
- data source types, 2-4
- data sources, 1-1, 3-4, 6-26
  - concatenating, 9-9 to 9-11
  - describing field relationships, 1-4
  - describing fields, 1-4, 4-2
  - describing files, 1-2 to 1-3
  - documenting, 1-11
  - FOCUS, 3-4
  - identifying, 2-1
  - joining, 6-26
  - partitioning, 6-26
  - relational, 3-2
  - rotating the structure, 3-31 to 3-32, 3-34
  - USE command, 9-1 to 9-3
- data storage, D-2, D-4
- data types, 4-12 to 4-13, 4-41 to 4-43, 6-18
  - alphanumeric, 4-20 to 4-21
  - date display options, 4-22
  - date storage, 4-23
  - dates, 4-21, 4-27 to 4-28, 4-32
  - date-time, 4-32 to 4-33, 4-38
  - floating-point double-precision, 4-14
  - floating-point single-precision, 4-15
  - integer, 4-14
  - internal representation, 6-18
  - numeric display options, 4-16, 4-19
  - packed-decimal, 4-15
  - text, 4-40
- database administration. *See* DBA
- database descriptions, 1-1 to 1-2
  - applications and, 1-3
  - creating, 1-4
  - field declarations, 1-4, 4-2
  - field relationships, 1-4
  - file declarations, 1-3
  - Master Files, 1-3, 1-9
- DATANAME attribute, 6-22 to 6-25
- DATASET attribute, 2-1, 2-5
  - fixed-format sequential data sources, 2-8
  - FOCUS data sources, 2-5 to 2-7
  - priority, 2-5
  - VSAM data sources, 2-9 to 2-10
- date calculations, 4-29
- date conversions, 4-29
- date data types, 4-21, 4-27 to 4-28, 4-32
  - calculations, 4-29
  - converting values, 4-29
  - Dialogue Manager and, 4-31
  - display options, 4-22
  - extract files and, 4-31
  - graph considerations, 4-31
  - internal representation, 4-30
  - literals, 4-24, 4-27 to 4-28
  - non-standard formats, 4-31
  - RECAP command and, 4-31
  - separators, 4-26
  - storage, 4-23
  - translation, 4-26
- date display options, 4-22
- date literals, 4-24, 4-27 to 4-28
- date separators, 4-26
- date translation, 4-26
- DATEDISPLAY parameter, 4-30
  - ALLOWCVTERR and, 4-31
- DATEFORMAT parameter, 4-38
- date-time data type, 4-32 to 4-33, 4-38
- DBA (database administration), 10-1
  - security, 10-1

- DBA attributes, 10-2
- DBA decision table, 10-20
  - displaying, 10-32 to 10-33
- DBA security, 10-6 to 10-8
  - HOLD files, 10-6 to 10-8
- DBAFILE attribute, 10-25 to 10-26
  - file naming requirements, 10-27
- DBATABLE command, 10-20, 10-32 to 10-33
- debugging Master Files, 8-3
- decimal data types, 4-14 to 4-15
- declarations in Master Files, 1-10
  - documenting, 1-11
  - improving readability, 1-10 to 1-11
- DECRYPT command, 10-15
  - user access to, 10-15
- DEFCENT attribute, 4-2, 8-2
  - CHECK FILE command and, 8-2, 8-8
  - date-time data type and, 4-38
- DEFINE attribute, 4-51 to 4-52, 8-11
  - CHECK HOLD option, 8-10
- DEFINE command, 10-15
  - rounding, D-9
  - user access to, 10-15
- DEFINITION attribute, 4-50
- delimiters, 5-45 to 5-47
- DESC attribute, 4-50
- descendant segments, 3-12, 6-2
  - FOCUS data sources, 6-2
- describing data sources, 1-1 to 1-2
  - Access Files, 1-2
  - field declarations, 1-4, 4-2
  - field relationships, 1-4
  - file declarations, 1-3
  - Master Files, 1-2
- DESCRIPTION attribute, 4-50
- designing FOCUS data sources, 6-2 to 6-3
- display formats for fields, 4-12 to 4-13
  - alphanumeric, 4-20 to 4-21
  - currency display options, 4-17 to 4-18
  - date display options, 4-22
  - date storage, 4-23
  - dates, 4-21, 4-27 to 4-28, 4-32
  - date-time, 4-32 to 4-33, 4-38
  - floating-point double-precision, 4-14
  - floating-point single-precision, 4-15
  - integer, 4-14
  - numeric display options, 4-16, 4-19
  - packed-decimal, 4-15
  - text, 4-40
- display options, 4-12 to 4-13
  - currency symbols, 4-17 to 4-18
  - date, 4-22
  - numeric, 4-16, 4-19
- documenting data sources, 1-11
- documenting fields, 4-50
- double-precision fields, D-3
  - rounding, D-3
- double-precision floating-point data type, 4-14
- DUMMY root segments, 5-27 to 5-30
- duplicate field names, 4-4 to 4-7
  - CHECK FILE command and, 8-2, 8-4
- DUPLICATE option to CHECK FILE command, 8-2, 8-4
- DYNAM ALLOCATE command, 1-5, 1-7 to 1-8
- DYNAM command, 1-9
- DYNAM FREE LONGNAME command, 1-7
- dynamic join structures, 7-15 to 7-16
  - comparing with static, 7-17

**E**

- edit options, 4-12 to 4-13
  - date, 4-22
  - numeric, 4-16, 4-19
- EDUCFILE data source, A-7
- EMPDATA data source, A-17
- EMPLOYEE data source, A-3
- ENCRYPT attribute, 10-31
- ENCRYPT command, 10-30
  - user access to, 10-15
- entry-sequenced data sources (ESDS), 5-1
- error checking of Master Files, 8-3
- error files, B-1
  - CMS, B-2
  - MVS, B-2
- error messages, B-1
  - displaying, B-3
- ESDS (entry-sequenced data sources), 5-1
- exits, C-1
  - FOCSAM interface, C-2
  - functional requirements, C-3
  - UACCT, 10-40
  - ZCOMP1, C-12
- EXPERSON data source, A-18
- external data sources, 8-4
  - checking Master Files, 8-4
- extract files, 8-2
  - CHECK FILE command and, 8-2, 8-8 to 8-10

**F**

- F data type, 4-15
- FDEFCENT attribute, 2-1, 8-2
  - CHECK FILE command and, 8-2, 8-8
  - date-time data type and, 4-38
- field aliases, 4-10 to 4-11
- FIELD attribute, 4-3
- field attributes, 6-15
  - FOCUS data sources, 6-15
- field formats, 4-12 to 4-13, 4-41 to 4-43, D-6
  - alphanumeric, 4-20 to 4-21
  - date display options, 4-22
  - date storage, 4-23
  - dates, 4-21, 4-27 to 4-28, 4-32
  - date-time, 4-32 to 4-33, 4-38
  - floating-point double-precision, 4-14
  - floating-point single-precision, 4-15
  - integer, 4-14
  - numeric display options, 4-16, 4-19
  - packed-decimal, 4-15
  - redefining, D-6 to D-7
  - text, 4-40
- field names, 4-3 to 4-7
  - CHECK FILE command and, 8-2, 8-4
  - qualified, 4-9
- FIELD option to RESTRICT attribute, 10-14
- field values, 10-20 to 10-21
  - restricting access to, 10-20 to 10-24
  - validating, 4-46 to 4-47
- FIELDNAME attribute, 4-3
- FIELDNAME parameter, 4-4

- fields, 1-2, 3-5, 4-1
  - defining groups, 3-2
  - describing, 1-4, 4-2
  - documenting, 4-50
  - filler, 3-6
  - key, 3-4
  - naming, 4-3 to 4-7
  - redefining, 5-20 to 5-21
  - repeating, 5-2, 5-9, 5-11, 5-35
  - restricting access to, 10-18 to 10-19
- FIELDTYPE attribute, 6-16, 6-18
- FILE attribute, 2-2
- file descriptions, 1-1 to 1-2
  - applications and, 1-3
  - creating, 1-4
  - field declarations, 1-4, 4-2
  - field relationships, 1-4
  - file declarations, 1-3
  - Master Files, 1-3, 1-9
- file security, 9-8
  - read-only access, 9-8
- file specifications, 9-5 to 9-6
  - changing with USE command, 9-5 to 9-6
- FILENAME attribute, 2-1 to 2-2
- files, 10-40
  - absolute integrity, 10-40 to 10-41
- FILESUFFIX attribute, 2-2 to 2-3
- filler fields, 3-6, 5-2
- filters, 10-28
  - data security, 10-28
- FINANCE data source, A-14
- FIND function, 2-5
  - DATASET attribute and, 2-5
- FIND option to ACCEPT attribute, 6-15
- fixed-format data sources, 5-2
  - generalized record types, 5-31 to 5-32
  - multiple record types, 5-22 to 5-24, 5-34 to 5-35
  - nested repeating fields, 5-15 to 5-16
  - order of repeating fields, 5-19
  - parallel repeating fields, 5-14, 5-16
  - position of repeating fields, 5-17 to 5-18
  - positionally related records, 5-24 to 5-25
  - repeating fields, 5-11 to 5-13, 5-35
  - unrelated records, 5-27 to 5-28
- floating-point double-precision data type, 4-14
- floating-point fields, 4-14
  - rounding, D-2 to D-3
- floating-point single-precision data type, 4-15
- FOC2GIGDB parameter, 6-19
- FOCSAM interface, C-2
  - Access Files, C-4
  - calling sequence, C-5
  - user exits, C-2
  - work area control block, C-6, C-8
  - ZCOMP1 user exit, C-12
- FOCUS data sources, 3-4, 6-1
  - ACCEPT attribute, 6-15
  - allocating, 6-28
  - changing, 6-4
  - data type representation, 6-18
  - designing, 6-2 to 6-3
  - field attributes, 6-15
  - FIND option, 6-15
  - FORMAT attribute, 6-18
  - INDEX attribute, 6-16, 6-18
  - internal storage lengths, 6-18
  - joining, 6-3
  - key fields, 6-7
  - LOCATION attribute, 6-9 to 6-12
  - MISSING attribute, 6-18
  - partitioning, 6-19 to 6-21, 6-24 to 6-25
  - rebuilding, 6-28
  - segment relationships, 6-9
  - segment sort order, 6-8
  - segments, 6-5
  - SEGTYPE attribute, 6-5, 6-7
  - sorting, 6-28
  - support for, 6-19

FOCUS data sources (*continued*)

USE command, 9-1 to 9-3

FORMAT attribute, 4-12 to 4-13

free-format data sources, 5-2, 5-5 to 5-6

repeating fields, 5-9 to 5-10

FSCAN command, 10-16

user access to, 10-16

FYRTHRESH attribute, 2-1, 8-2

CHECK FILE command and, 8-2, 8-8

date-time data type and, 4-38

## G

generalized record types, 5-31 to 5-32

GETPRV exit, C-2

GGDEMOG data source, A-29

GGORDER data source, A-30

GGPRODS data source, A-31

GGSALES data source, A-32

GGSTORES data source, A-33

Gotham Grinds data sources, A-29

GGDEMOG, A-29

GGORDER, A-30

GGPRODS, A-31

GGSALES, A-32

GGSTORES, A-33

group keys, 5-1, 5-7 to 5-9

groups of fields, 3-2

## H

H data type, 4-32 to 4-33, 4-38

help PF keys, 4-48

HELPMESSAGE attribute, 4-47

CHECK HOLD and, 8-10

HLI (Host Language Interface), 10-15

user access to, 10-15

HOLD ALL option to CHECK FILE command,  
8-2, 8-8 to 8-10

HOLD files, 10-6 to 10-8

DBA security, 10-6 to 10-8

HOLD option to CHECK FILE command, 8-2, 8-8  
to 8-10

HOLDSTAT files, 10-6 to 10-8

DBA security, 10-6 to 10-8

Host Language Interface (HLI), 10-15

user access to, 10-15

host segments, 3-9, 7-19

## I

I data type, 4-14

IDCAMS utility, 5-42, 5-44

identifying fields, 3-4

INDEX attribute, 6-16, 6-18

joining data sources, 6-17

index buffers for VSAM data sources, 5-41

indexed data sources, 3-21

instances, 3-3, 3-8

integer data type, 4-14

integer fields, D-3

rounding, D-2 to D-3

internal representation of data types, 6-18

internal representation of dates, 4-30

ISAM data sources, 5-1

describing, 5-7

generalized record types, 5-31 to 5-32

group keys, 5-7 to 5-9

multiple record types, 5-22 to 5-24, 5-34 to 5-35

nested repeating fields, 5-15

order of repeating fields, 5-19

parallel repeating fields, 5-14

position of repeating fields, 5-17 to 5-18

positionally related records, 5-24 to 5-25

repeating fields, 5-11 to 5-13, 5-35

unrelated records, 5-27 to 5-28

## J

JOBFILE data source, A-6

JOIN command, 7-17

data security, 10-25 to 10-26, 10-28

long field names and, 4-5

join structures, 3-8, 7-1

between different file types, 3-30

comparing, 7-17

cross-referenced segments, 3-9, 7-10, 7-11, 7-13, 7-19

defined in Master File, 7-3, 7-15 to 7-16

dynamic, 7-15 to 7-16

host segments, 3-9

linked segments, 7-10 to 7-11, 7-13

many-to-many relationships, 3-22 to 3-23

non-unique, 7-3, 7-7 to 7-8

recursive, 3-27 to 3-28, 7-23

static, 7-3, 7-17

types, 7-2

unique, 7-3, 7-5 to 7-6

joining data sources, 6-26

FOCUS, 6-3

INDEX attribute, 6-17

partitioned, 6-27

## K

key fields, 3-4, 6-7

FOCUS data sources, 6-7

key-sequenced data sources (KSDS), 5-1

KSDS (key-sequenced data sources), 5-1

## L

leaf segments, 3-9, 3-12

LEDGER data source, A-13

linked segments, 7-10 to 7-11, 7-13

hierarchies, 7-14

literals for dates, 4-24, 4-27 to 4-28

load procedures, A-1 to A-2

LOCATION attribute, 6-9 to 6-12, 6-22 to 6-25, 9-14

location files, 6-12

logical dependence, 3-9

logical independence, 3-13

logical views, 3-5

long alphanumeric fields, 4-21

long field names, 4-4 to 4-7

alternate file views and, 4-5

indexed fields and, 4-5

temporary fields, 4-51

long names, 1-5

Master Files, 1-5 to 1-9

LONGNAME option, 1-5, 1-7 to 1-9

## M

MAINTAIN command, 10-15

user access to, 10-15

many-to-many relationships, 3-22 to 3-23

MAPFIELD alias, 5-38 to 5-41

MAPVALUE fields, 5-38 to 5-41

Master Files, 1-2 to 1-3, 1-9, 3-2, 10-16

allocating data sources, 2-5

applications and, 1-3

changing, 8-1

changing names, 9-5 to 9-6

changing names with USE command, 9-5

checking, 8-1, 8-3

common errors, 8-4

creating, 1-4

creating filler fields, 3-6

declarations, 1-10

defining joins, 7-3, 7-15 to 7-16

diagrams, 8-5, 8-7

documenting, 1-11

encrypting, 10-30

for FOCSSAM interface, C-4

identifying parent segments, 3-8 to 3-9

improving readability, 1-10 to 1-11

long names, 1-5 to 1-9

**Master Files** (*continued*)

- names of data sources, 2-2
  - naming, 1-5
  - one-to-many relationships, 3-21
  - samples, A-1 to A-2
  - security attributes, 10-2
  - segment relationships, 3-9
  - segments, 3-3
  - statistics, 8-2, 8-8
  - types of data sources, 2-2 to 2-3
  - user access to, 10-16
  - validating, 1-11
- MASTERNAME** attribute, 6-22 to 6-25
- MISSING** attribute, 4-44 to 4-45
  - ALLOWCVTERR** and, 4-31
- missing values, 4-44 to 4-45, 5-47
- MODIFY** command, 10-15
  - user access to, 10-15
- MOVIES** data source, A-24
- multi-path data sources, 3-14
- multiple groups of fields, 3-2
- multiple paths, 3-13 to 3-14
- multiple record types, 3-21, 5-22 to 5-24, 5-34 to 5-35, 5-37
- multiple segments, 3-9, 3-11
- multiply occurring fields, 5-2, 5-9 to 5-13, 5-35, 5-37
  - MAPFIELD** and **MAPVALUE**, 5-38 to 5-41
    - nested, 5-15 to 5-16
    - order, 5-19
    - parallel, 5-14, 5-16
    - position, 5-17 to 5-18
    - record length, 5-21
- multi-segment records, 3-7
- multi-threads, 9-13
  - USE** command, 9-13

**N**

- naming conventions, 1-5
    - Master Files, 1-5
  - naming fields, 4-3 to 4-7
  - National Language Support (NLS), 1-10
  - nested repeating fields, 5-15 to 5-16
  - NLS (National Language Support), 1-10
  - non-FOCUS data sources, 8-4
    - checking Master Files, 8-4
  - non-unique join structures, 7-7
    - specifying, 7-8
  - NOPRINT** option to **RESTRICT** attribute, 10-14
  - null values, 4-44 to 4-45
  - numbers, D-1
    - rounding, D-1
  - numeric data types, 4-13
    - display options, 4-16, 4-19
    - floating-point double-precision, 4-14
    - floating-point single-precision, 4-15
    - integer, 4-14
    - packed-decimal, 4-15
  - numeric display options, 4-16, 4-19
  - numeric fields, D-1
- O**
- OCCURS** attribute, 3-21, 5-12 to 5-13, 5-16
  - one-to-many relationships, 3-19 to 3-20
  - one-to-many segment relationships, 6-2
    - FOCUS** data sources, 6-2, 6-9
  - one-to-one relationships, 3-16 to 3-17
  - one-to-one segment relationships, 6-2
    - FOCUS** data sources, 6-2, 6-9
  - online help, 4-47
    - displaying, 4-47 to 4-48
  - ORDER** fields, 5-19



- P**
- P data type, 4-15
  - packed-decimal data type, 4-15
  - packed-decimal fields, 4-15
    - rounding, D-2, D-4
  - parallel repeating fields, 5-14, 5-16
  - PARENT attribute, 3-8 to 3-9
  - parent segments, 3-8 to 3-9, 3-13
  - parent-child relationships, 3-9 to 3-10
  - partitioned data sources, 6-26, 6-28
    - joining, 6-27
  - partitioning FOCUS data sources, 6-19 to 6-21, 6-24 to 6-25
  - PASS command, 10-9 to 10-10
  - PASS command, 10-35
  - passwords, 10-6
    - changing, 10-6
    - setting externally, 10-34
    - suppressing display, 10-35
    - variable, 10-35
  - paths, 3-15, 3-31
    - multiple, 3-14
    - single, 3-13
  - paths in data sources, 6-2
    - FOCUS data sources, 6-2
  - PAYHIST data source, A-20
  - PF keys for help, 4-48
  - physical relationships, 3-7
  - PICTURE option to CHECK FILE command, 8-2, 8-5, 8-7
  - POSITION attribute, 3-21, 5-17 to 5-18
  - positionally related records, 5-24 to 5-26
  - PRIVATE suffix for FOCSAM interface, C-4
  - procedures, 10-34
    - encrypting, 10-36
    - load, A-1 to A-2
    - security, 10-34
  - PROD data source, A-10
  - program accounting, 10-1, 10-37
    - activating, 10-38
  - PROGRAM option to RESTRICT attribute, 10-37
- Q**
- QUALCHAR parameter, 4-5
  - qualification character, 4-5
  - qualified field names, 4-4 to 4-7, 4-9
    - levels of qualification, 4-9
    - temporary fields, 4-52
- R**
- read/write access, 10-12
  - reading non-FOCUS data sources, 5-48
  - read-only access, 9-8, 10-12
  - REBUILD command, 10-16
    - user access to, 10-16
  - record length in sequential data sources, 5-21
  - record types, 5-2
    - generalized, 5-31 to 5-32
    - multiple, 5-22 to 5-24, 5-34 to 5-35, 5-37
  - RECTYPE fields, 3-21 to 5-24, 5-31 to 5-32, 5-34 to 5-35, 5-37
    - MAPFIELD and MAPVALUE, 5-38 to 5-41
  - recursive join structures, 3-27 to 3-28, 7-23
  - redefining field formats, D-6 to D-7
  - redefining fields in non-FOCUS data sources, 5-20 to 5-21
  - REGION data source, A-15

relational data sources, 3-5  
 logical views, 3-5  
 omitting fields, 3-6

repeating fields, 5-2, 5-9 to 5-13, 5-35, 5-37  
 MAPFIELD and MAPVALUE, 5-38 to 5-41  
 nested, 5-15 to 5-16  
 order, 5-19  
 parallel, 5-14, 5-16  
 position, 5-17 to 5-18  
 record length, 5-21

resource limitation, 10-37

resource management, 10-37, 10-39

RESTRICT attribute, 10-17  
 options, 10-14  
 PROGRAM, 10-37

RESTRICT command, 10-32  
 user access to, 10-16

restricting access, 10-18 to 10-19  
 to fields, 10-18 to 10-19  
 to segments, 10-18 to 10-19

RETRIEVE option to CHECK FILE command, 8-2

root segments, 3-12, 6-2  
 alternate, 3-31 to 3-32, 3-34  
 FOCUS data sources, 6-2

rounding numbers, D-1  
 COMPUTE fields, D-9  
 DEFINE fields, D-9  
 double-precision fields, D-3  
 floating-point fields, D-2 to D-3  
 in calculations, D-6  
 integer fields, D-2 to D-3  
 packed-decimal fields, D-2, D-4

## S

SALES data source, A-8 to A-9

SAME option to RESTRICT attribute, 10-14

sample data sources, A-2  
 CAR, A-11  
 COMASTER, A-21  
 COURSES, A-16  
 creating, A-1 to A-2  
 EDUCFILE, A-7  
 EMPDATA, A-17  
 EMPLOYEE, A-3  
 EXPERSON, A-18  
 FINANCE, A-14  
 Gotham Grinds data sources, A-29  
 JOBFIL, A-6  
 LEDGER, A-13  
 MOVIES, A-24  
 PAYHIST, A-20  
 PROD, A-10  
 REGION, A-15  
 SALES, A-8 to A-9  
 TRAINING, A-19  
 VIDEOTR2, A-26  
 VideoTrk, A-24

SCAN command, 10-16  
 user access to, 10-16

security, 10-1, 10-3 to 10-4  
 access levels, 10-12, 10-17  
 encrypting data segments, 10-31  
 encrypting Master Files, 10-30  
 encrypting procedures, 10-36  
 FOCUS procedures, 10-34  
 FOCUSID routine, 10-34  
 identifying users, 10-9  
 locking users out of FOCUS, 10-36  
 passwords, 10-10, 10-34  
 program accounting, 10-37  
 read-only access, 9-8  
 RESTRICT command, 10-32  
 storing DBA information centrally, 10-25 to 10-26  
 UACCT exit routine, 10-40  
 user program specifications, 10-38

- security attributes, 10-2, 10-29
  - ACCESS, 10-12
  - DBA, 10-5
  - DBAFILE, 10-25 to 10-26
  - RESTRICT, 10-17
  - USER, 10-9
- segment chains, 3-3
- segment instances, 3-3
- SEGMENT option to RESTRICT attribute, 10-14
- segments, 1-4, 3-2
  - ancestral, 3-13
  - child, 3-8
  - cross-referenced, 3-9
  - descendant, 3-12
  - encrypting, 10-31
  - host, 3-9
  - key fields, 3-4
  - leaf, 3-12
  - logical views, 3-5
  - naming, 3-4 to 3-5
  - one-to-one relationships, 3-16
  - parent, 3-8 to 3-9
  - restricting access to, 10-18 to 10-19
  - root, 3-12
  - storing, 6-10
  - timestamping, 6-12 to 6-14
  - unique, 3-16
- SEGNAME attribute, 3-3
  - VSAM and ISAM considerations, 5-7
- SEGTYPE attribute, 3-3, 3-9, 8-5
  - displaying, 8-5
  - FOCUS data sources, 6-5, 6-7 to 6-8
  - sequential data source considerations, 5-6
  - VSAM considerations, 5-7
- separators for dates, 4-26
- sequential data sources, 5-1 to 5-2, 5-4 to 5-5, 5-45 to 5-46
  - describing, 5-6
  - fixed format, 5-2
  - free format, 5-5 to 5-6
  - generalized record types, 5-31 to 5-32
  - multiple record types, 5-22 to 5-24, 5-34 to 5-35
  - multiply occurring fields, 5-21
- sequential data sources (*continued*)
  - nested repeating fields, 5-15 to 5-16
  - order of repeating fields, 5-19
  - parallel repeating fields, 5-14, 5-16
  - position of repeating fields, 5-17 to 5-18
  - positionally related records, 5-24 to 5-25
  - record length, 5-21
  - repeating fields, 5-9 to 5-13, 5-35
  - unrelated records, 5-27 to 5-28
- SET parameters, 6-3
  - ACCBLN, 4-46
  - ACCEPTBLANK, 4-46
  - ALLOWCVTERR, 4-31
  - AUTOPATH, 6-3
  - DATEDISPLAY, 4-30
  - DATEFORMAT, 4-38
  - FIELDNAME, 4-4
  - FOC2GIGDB, 6-19
  - PASS, 10-9 to 10-10
  - QUALCHAR, 4-5
  - TRANTERM, 4-17
  - USER, 10-9 to 10-10
- shadow pages, 10-40 to 10-41
  - implementing, 10-41
- Simultaneous Usage, 9-12
  - USE command, 9-12
- single-precision floating-point data type, 4-15
- smart dates, 4-23
- specifying an Access File in a Master File, 6-21
- SQL Translator and long field names, 4-5
- static join structures, 7-3
  - comparing with dynamic, 7-17
- storage of date data types, 4-23
- structure diagrams, A-1 to A-2
- subroutines, C-10
  - user-coded for data access, C-10
- SUFFIX attribute, 2-1 to 2-3, C-4
  - PRIVATE, C-4
  - values, 2-4
  - VSAM and ISAM, 5-7

- SUFFIX=COM attribute, 5-4
- SUFFIX=COMT attribute, 5-4
- SUFFIX=TABT attribute, 5-5
- ## T
- tab-delimited data sources, 5-2, 5-4 to 5-5
- TABLE command, 10-16  
user access to, 10-16
- TAG attribute and CHECK HOLD, 8-10
- temporary fields, 4-51  
creating, 4-51 to 4-52  
long field names and, 4-5, 4-51  
qualified field names, 4-52
- TRANTERM parameter, 4-17
- text data type, 4-40  
long field names and, 4-5
- text fields, 6-11  
storing, 6-11
- time stamps, 6-12
- timestamp data type, 4-32 to 4-33, 4-38
- TITLE attribute, 4-49  
CHECK HOLD and, 8-10
- token-delimited data sources, 5-2, 5-45 to 5-47
- TRAINING data source, A-19
- translation of dates, 4-26
- TX data type, 4-40
- ## U
- UACCT exit routine, 10-40
- unique join structures, 7-3, 7-5 to 7-6  
decoding with, 7-6
- unique segments, 3-16
- unrelated records, 5-27 to 5-30
- update access, 10-12
- usage accounting of resources, 10-40
- USAGE attribute, 4-12 to 4-13
- USAGE format, 4-42 to 4-43, 5-45
- USE command, 9-1 to 9-4  
alternate file specifications, 9-5 to 9-6  
clearing, 9-4  
data source concatenation, 9-9 to 9-11  
displaying options, 9-14  
file modes, 9-5 to 9-6  
file names, 9-5 to 9-6  
file types, 9-5 to 9-6  
multiple data sources, 9-4  
multi-threads, 9-13  
new data sources, 9-6 to 9-7  
read-only access, 9-8  
Simultaneous Usage, 9-12
- USER attribute, 10-9
- user exits, C-1  
FOCSAM interface, C-2  
functional requirements, C-3  
ZCOMP1, C-12
- user programs, 10-38  
activating, 10-38  
specifications, 10-38
- user-coded data access routines, C-10
- user-written procedures, 5-48
- user-written subroutines, 10-37  
program accounting, 10-37
- ## V
- validating field values, 4-46 to 4-47
- validating Master Files, 1-11
- VALUE option to RESTRICT attribute, 10-14,  
10-20 to 10-24
- variables, 10-35  
in passwords, 10-35
- VIDEOTR2 data source, A-26
- VideoTrk data source, A-24

views, 3-5

virtual fields, 4-51

  creating, 4-51 to 4-52

  long field names and, 4-51

  qualified field names, 4-52

virtual fields in Master Files and CHECK HOLD,  
8-11

VSAM data sources, 5-1

  alternate indexes, 5-42 to 5-44

  data buffers, 5-41

  describing, 5-7

  FOCSAM interface user exit, C-2

  generalized record types, 5-31 to 5-32

  group keys, 5-7 to 5-9

  IDCAMS utility, 5-42, 5-44

  index buffers, 5-41

  multiple record types, 5-22 to 5-24, 5-34 to 5-35,  
  5-37

  nested repeating fields, 5-15 to 5-16

  order of repeating fields, 5-19

  parallel repeating fields, 5-14, 5-16

  position of repeating fields, 5-17 to 5-18

  positionally related records, 5-24 to 5-26

  repeating fields, 5-11 to 5-13, 5-35, 5-37

  unrelated records, 5-27 to 5-30

VSAM data sources (*continued*)

  ZCOMP1 parameters, C-13

  ZCOMP1 user exit, C-12

## W

WHERE attribute, 6-22 to 6-25

work area control block, C-6, C-8

write-only access, 10-12

## Y

Y2K attributes in Master Files, 4-2

  date-time data type, 4-38

Year 2000 attributes in Master Files, 4-2

  date-time data type, 4-38

YRTHRESH attribute, 4-2, 8-2

  CHECK FILE command and, 8-2, 8-8

  date-time data type and, 4-38

## Z

ZCOMP1 user exit, C-12

  linking, C-12

  parameters, C-13

---

# 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](mailto:books_info@ibi.com)

**Web form:** <http://www.informationbuilders.com/bookstore/derf.html>

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

Telephone: \_\_\_\_\_ Date: \_\_\_\_\_

E-mail: \_\_\_\_\_

Comments:

---

# Reader Comments