

Fusion

User's Manual
for EDA 4.3 and FOCUS 7.1

Contents

1	Introducing Fusion	1-1
	Overview of Traditional Systems	1-2
	Benefits and Limitations of Traditional Systems	1-2
	Multi-Dimensional Data Sources	1-4
	Fusion	1-6
	Fusion Data Sources	1-7
	The Fusion Schema	1-7
	High-Performance Features	1-9
	Database Administration Tools	1-11
2	Understanding Your Fusion Schema	2-1
	The Fusion Schema	2-2
	Fusion Declarations	2-2
	The Fusion Master File	2-3
	File Attributes	2-5
	Assigning a Logical Name to a File: FILENAME	2-6
	Identifying the Type of Data Source: FILETYPE	2-6
	Identifying a Fusion Access File: ACCESSFILE	2-7
	Identifying a Repository for a Data Source: DATASET	2-7
	Supplying Descriptive Text: REMARKS	2-8
	Defining the Correct Century for a File: FDEFCENT AND FYRTHRESH	2-8
	Class Attributes	2-9
	Assigning a Name to a Class: CLASSNAME	2-11
	Indicating the Sort Sequence and Number of Fields in the Key: CLASSKEYS	2-11
	Identifying the Target Class: TARGET_OF	2-12
	Assigning a Logical Name to a File: LOCATION	2-13
	Describing Class Relationships: SUBCLASS_OF and PART_OF	2-13
	Vertical Partitioning: SECTION	2-14
	Field Attributes	2-15
	Assigning a Name to a Field: FIELDNAME	2-18
	Assigning an Alternate Name to a Field: ALIAS	2-19
	Describing a Field's Data Type and the Display of the Field Value: USAGE	2-19
	Defining the Format of a Field: ACTUAL	2-20
	Defining the Correct Century for a Field: DEFCENT AND YRTHRESH	2-21
	Supplying a Column Title for a Field on a Report: TITLE	2-22
	Limiting the Number of Unique Values for a Field: MAXVALUES	2-22
	Missing Data Support: MISSING	2-23
	Identifying Acceptable Values for a Field: ACCEPTVALUES	2-23
	Supplying Text to Display When a Value Fails a Validation Test: HELPMESSAGE	2-24
	Providing a Field Description: DESCRIPTION	2-24
	Setting Index Attributes: INDEX	2-25
	Describing a Group of Fields	2-27
	Joining Classes: JOIN_TO, JOIN_TO ALL, and LINK_TO	2-28

The Fusion Access File	2-31
File Attributes.....	2-32
Describing a Data Source: MASTERNAME and DATANAME.....	2-35
Describing a Separately Stored Section or Class: LOCATION	2-36
Describing a Horizontal Partition: WHERE.....	2-37
Assigning a Logical Name to an Index: INDEXLOCATION and MDILOCATION	2-39
Describing Joined Files	2-40
3 Using Fusion’s High-Performance Features	3-1
Creating a Multi-Dimensional Index	3-2
Choosing an Indexing Strategy	3-2
Choosing Dimensions for Your Index.....	3-2
Using a Multi-Dimensional Index With a Query	3-3
Querying a Multi-Dimensional Index.....	3-4
Using AUTOINDEX to Choose a Multi-Dimensional Index.....	3-6
Joining to a Multi-Dimensional Index	3-7
Choosing Source Fields.....	3-8
Encoding Values in a Multi-Dimensional Index	3-11
Creating Horizontal and Vertical Partitions	3-13
Using a Horizontal Partition.....	3-13
Using a Vertical Partition	3-15
4 Database Administration Facilities.....	4-1
Populating a Fusion Data Source: MODIFY FIXFORM	4-2
Migrating Data to Fusion: HOLD FORMAT FUSION.....	4-3
Migrating a Master File to a Fusion Master File: REBUILD MIGRATE.....	4-6
Migrating a FOCUS Data Source to Fusion: REBUILD MIGRATE.....	4-7
Writing a New Partition in a Fusion Access File: HOLD FORMAT FUSION.....	4-11
Building and Maintaining a Multi-Dimensional Index.....	4-14
Partitioning a Multi-Dimensional Index.....	4-17
Querying the Progress of a Multi-Dimensional Index.....	4-18
Displaying a Warning Message.....	4-18
Retrieving Data Directly From a Multi-Dimensional Index.....	4-19
Providing Information About a Fusion Data Source	4-19
Using a Non-Intrusive External Index	4-20
Rebuilding Data Sources Larger Than Two Gigabytes	4-25
Using SmartLoad to Update Multiple Partitions	4-26

5	Creating Derived Fields	5-1
	Using a Derived Field.....	5-2
	Calculating a Derived Field With Missing Values	5-3
	Types of Expressions.....	5-4
	Expressions and Field Formats.....	5-5
	Numeric Expressions.....	5-5
	Arithmetic Operators.....	5-6
	Order of Evaluation	5-6
	Date Expressions	5-7
	Field Formats for Date Values	5-7
	Performing Calculations on Dates.....	5-8
	Selecting the Format of the Result Field.....	5-8
	Working With Dates in Date Format.....	5-9
	Alphanumeric Expressions	5-10
	Concatenating Character Strings	5-11
	Logical Expressions.....	5-11
	Relational Expressions	5-12
	Boolean Expressions	5-13
	Conditional Expressions	5-13
6	Fusion Security	6-1
	Implementing Database Security.....	6-2
	Identifying the Database Administrator	6-3
	Identifying a User.....	6-5
	Specifying the Type of Access for a User	6-7
	Limiting a User's Access Within a File	6-8
	Combining RESTRICT and ACCESS Attributes to Limit Access	6-10
	Storing Security Information in a Central File	6-11
	Encrypting a Master File	6-15
	Stored Procedure Security	6-15
7	The Fusion Catalog.....	7-1
	The Fusion Catalog Master File	7-2
	Describing the Fusion Catalog Master File	7-2
	Querying the Fusion Catalog Master File.....	7-6
A	Sample Schemas	A-1
	Master Files for the XMDII Multi-Dimensional Index.....	A-2
	The ORDERS Data Source	A-4

B	Fusion Functions and Subroutines.....	B-1
	Calling a Function or Subroutine.....	B-2
	Numeric Functions and Subroutines.....	B-3
	Performing Basic Numeric Calculations	B-3
	Performing Complex Numeric Manipulations	B-3
	Changing the Format of a Numeric Value.....	B-4
	Date Functions and Subroutines	B-4
	Calculating the Difference Between Dates.....	B-4
	Performing Arithmetic Calculations on Dates.....	B-5
	Alphanumeric Subroutines	B-6
C	Using Data Types and Display Options	C-1
	Numeric Format	C-2
	Numeric Display Options	C-5
	Rounding Values	C-7
	Alphanumeric Format.....	C-8
	Date Format.....	C-8
	Date Display Options	C-9
	Natural Date Literals and Numeric Date Literals	C-12
	How Date Fields Are Represented in Fusion	C-14
	Date Format Support	C-15
	Representing a Date as an Alphanumeric, Integer, or Packed Decimal	C-15
	Index	I-1

Cactus, EDA, FIDEL, FOCCALC, FOCUS, FOCUS Fusion, Information Builders, the Information Builders logo, SmartMode, SNAPpack, TableTalk, and Web390 are registered trademarks and Parlay, SiteAnalyzer, SmartMart, WebFOCUS, and WorldMART are trademarks of Information Builders, Inc.

Acrobat and Adobe are registered trademarks of Adobe Systems Incorporated.

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.

Intel is a registered trademark of Intel Corporation.

ACF/VTAM, AIX, AS/400, CICS, DB2, DRDA, Distributed Relational Database Architecture, IBM, MQSeries, MVS, OS/2, OS/400, RACF, RS/6000, S/390, VM/ESA, and VTAM are registered trademarks and DB2/2, Hiperspace, IMS, MVS/ESA, QMF, SQL/DS, VM/XA and WebSphere 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.

NetWare and Novell are registered trademarks of Novell, Inc.

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, JavaScript, NetDynamics, Solaris, SunOS, and iPlanet are trademarks of Sun Microsystems, Inc.

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

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

Allaire and JRun are trademarks of Allaire Corporation.

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 © 2000, 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 use SmartMart Fusion. It is intended for application developers and administrators who need to create high-performance multi-dimensional data sources.

Fusion is a high-performance data source with several multi-dimensional features that fully integrates with the EDA Server client/server architecture and mainframe FOCUS. With Fusion and EDA Server installed, you can create high-performance multi-dimensional data sources and access them with any front-end tool accessible to EDA Server, such as WebFOCUS, FOCUS Desktop, FOCUS/EIS, or any ODBC™-enabled product, such as Microsoft® Excel.

This manual emphasizes Fusion/DB, the engine side of Fusion. If you are not familiar with the TABLE query command and the MODIFY data maintenance command, refer to the *FOCUS Services* chapter in your EDA Server manual and to the *Fusion Creating Reports Manual*.

How This Manual Is Organized

This manual contains the following topics:

Chapter/Appendix		Description
1	<i>Introducing Fusion</i>	Provides an overview of multi-dimensional data source model and explains the specific features available in the Fusion high-performance data source.
2	<i>Understanding Your Fusion Schema</i>	Describes how to create a Master File and Access File, and how to add functionality to a data source.
3	<i>Using Fusion's High-Performance Features</i>	Explains how to choose multi-dimensional index dimensions, create smart horizontal and vertical partitions, and join to a multi-dimensional index.
4	<i>Database Administration Facilities</i>	Explains how to work with a data source, a multi-dimensional index, and the SmartLoad procedure.
5	<i>Creating Derived Fields</i>	Describes how to write expressions that calculate new dimensions or measures.
6	<i>Fusion Security</i>	Discusses the attributes of database security.

Chapter/Appendix		Description
7	<i>The Fusion Catalog</i>	Explains how to query the FUSCAT Master File for information about your Fusion schemas.
A	<i>Sample Schemas</i>	Lists Fusion schemas for the sample data sources used in examples throughout this manual.
B	<i>Fusion Functions and Subroutines</i>	Lists the functions and subroutines available for manipulating numeric, date, and alphanumeric values.
C	<i>Using Data Types and Display Options</i>	Describes data type support in Fusion as well as options available for displaying each data type in reports.

Documentation Conventions

The following conventions apply throughout this manual:

Convention	Description
THIS TYPEFACE or <i>this typeface</i>	Denotes syntax that you must enter exactly as shown.
<i>this typeface</i>	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
<u>underscore</u>	Indicates a default setting.
<i>this typeface</i>	Represents a placeholder (or variable) in a text paragraph, indicates a cross-reference, or emphasizes an important term.
this typeface	Highlights file names and commands (in a text paragraph) that must be lowercase.
this typeface	Indicates buttons, menu items, and dialog box options you can click or select.
Key + Key	Indicates keys that must be pressed simultaneously.
{ }	Indicates two choices from which you must choose one. You type one of these choices, not the braces.
[]	Indicates a group of optional parameters. None are required, but you may select one of them. Type only the information within the brackets, not the brackets.

Convention	Description
	Separates two mutually exclusive choices in a syntax line. You type one of these choices, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the parameters, not the ellipsis points (...).
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

See the *Information Builders Technical 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 Fusion?

Call Information Builders Customer Support Service (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your Fusion 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 also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

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

Information You Should Have

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

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

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

User Feedback

In an effort to produce effective documentation, the Documentation Services staff 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.

CHAPTER 1

Introducing Fusion

Topics:

- Overview of Traditional Systems
- Multi-Dimensional Data Sources
- Fusion

This topic describes the multi-dimensional data source model and the Fusion high-performance data source.

Overview of Traditional Systems

This topic describes how a traditional model functions in OLAP (Online Analytical Processing), OLTP (Online Transaction Processing), EIS (Executive Information Systems), and DSS (Decision Support Systems) applications.

Although computers have become increasingly accessible to business analysts in recent years, the full spectrum of enterprise data has remained largely inaccessible to them. This discrepancy results from the nature of traditional computer systems that are designed for high-speed transaction processing, not for ad hoc DSS, EIS, and OLAP queries.

This topic briefly describes the goals of traditional transaction processing systems, and the problems they present for business analysts. Multi-dimensional data sources, coexisting with and complementing operational systems, can circumvent these problems.

Benefits and Limitations of Traditional Systems

Traditionally, an organization builds systems geared to OLTP applications. Each system is designed around a specific application, such as an airline reservation system or a bank ATM system. In most cases, the goals of such systems are to maximize the speed of transaction processing and to keep the data source current. However, business analysts make decisions about the future of an organization by analyzing trends and summarizing data at various levels. While traditional computer systems meet some of the goals they were intended for, they also impose some limitations on OLAP, EIS, and DSS applications. The following elements of traditional computer systems illustrate this point.

Normalized Data

Benefits: In order to avoid anomalies in the data source that can result from storing the same data in more than one place, most redundant data is eliminated through a technique called normalization. This normalization process splits the data into a series of tables, each of which contains one discrete part of the whole data source.

In a normalized data source, most requests require information from more than one table. In order to locate related information between tables, a limited amount of data from one table must be duplicated in its related tables, introducing a certain amount of redundancy.

Limitations: Normalized data is non-intuitive and hard for non-programmers to understand.

B-tree Indexes

Benefits: An index is a list of each value of a field along with a pointer to its location in the data source. Just as you use an index in a book to locate specific information without reading the entire book, a Database Management System (DBMS) uses an index to locate specific information without reading the entire data source.

Each table normally has a key. A key is a field or a group of fields that uniquely identifies each row of a table. The key to a table usually has an index, and you can create indexes on other fields as needed.

Limitations: Table-specific indexes are of limited use in requests that join multiple tables. Also, in traditional systems, each index is limited to one field (or group of fields) within one table.

Joins

Benefits: Most requests against a normalized data source require data from multiple tables. To gather all the necessary information, the relevant tables must be joined.

Limitations: Joins between normalized tables generate significant processing overhead and require complex programming syntax.

High-volume, High-priority Transaction Processing

Benefits: A transaction in a traditional system, such as an airline reservation or ATM transaction, needs immediate access to the data source.

Limitations: A business analysis query requires timely access to the data source, but they may be locked out of the data source while high-priority transaction processing applications execute.

Current Data

Benefits: Once a transaction is processed, the data source must reflect the new data as soon as possible. If a flight becomes filled in an airline reservation system, the data source should not indicate that seats are still available. If a customer withdraws money from an account, his balance should reflect the deduction immediately.

Limitations: Business analysis requires access to historical data, while transaction-processing data sources contain mostly current data.

Other Limitations of Traditional Systems

- Business analysis requires access to both summarized and detail data. Calculating totals at run time slows processing; however, traditional data sources typically hold only detail data.
- Business analysis queries may need to pull together information from multiple systems, on different platforms, stored at separate locations.
- Business analysts ask for aggregations of data and other calculations that are beyond the ability of the data source to process.

Multi-Dimensional Data Sources

Since business analysis users have a different set of requirements than transaction processing users; they need systems and data sources that provide:

- Intuitive ways of organizing and relating data.
- Storage of aggregated data for efficient access in a query.
- Drill-down capabilities from the highest aggregate (or summary) level to lowest base (or detail) level, or vice versa.
- Efficient ad hoc query performance.
- No competition with transaction processing systems for data source access.
- Transparent access to historical as well as current data. Adding data for new time periods should not disrupt query access to existing data.
- Transparent and efficient access to multiple data sources from separate systems and locations.

A system to satisfy these requirements needs a new perspective on data that is provided by a multi-dimensional data source.

A multi-dimensional data source distinguishes between two types of information:

- **Dimensions.** Categories of interest, such as region, quarter, and department.
- **Facts or measures.** Data values for analysis, such as units sold and cost.

A *multi-dimensional index* (MDI) uses dimensions and all of their hierarchical relationships to point to specific facts in a Fusion data source.

Example

Using a Multi-Dimensional Data Source

Business analysis applications often need to locate entire categories of records rather than specific records based on a key. For example, an analyst for a food supply company may want to look at total sales of snack foods in the Northeast region over the past five years.

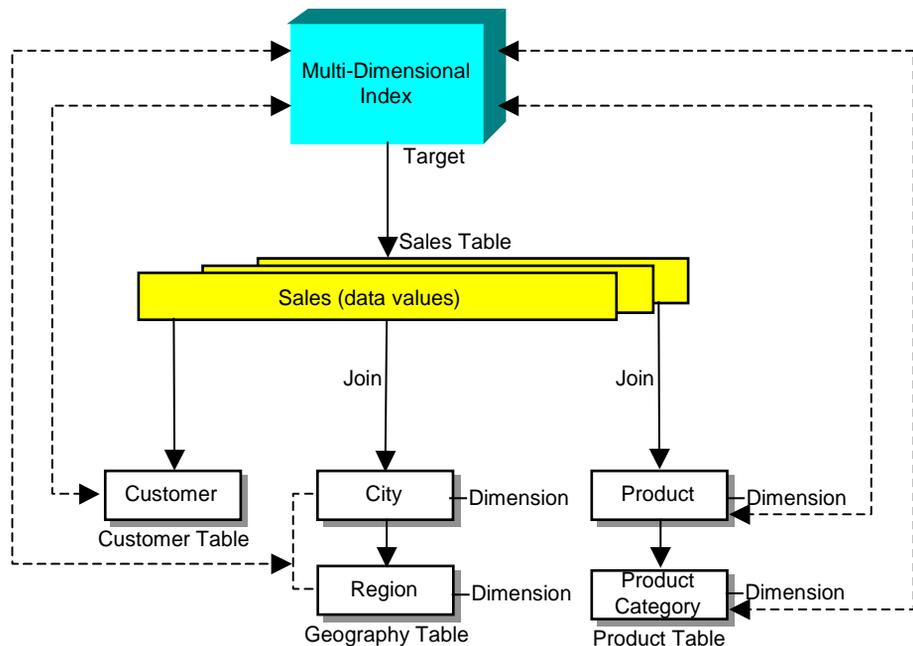
Consider the data source for the food supply company. It includes:

- A PRODUCTS table that lists each product sold by the company. Each product has a unique Product Code that identifies it; for example, the record for cheese-flavored popcorn has the product code SPC0001, while the record for whole-wheat pretzels has the product code SPZ0010. Product Code is the key to this table.
- A STATES table that lists each state in which the company does business. The State Code, such as NY or NJ, identifies each record in this table and is the key to this table.

- A fact table, SALES, that lists the number of units sold by product, state, year, and salesperson. Many records in this table apply to the same Product Code and State Code. In this table, Product Code and State Code are not keys, they are dimensions.

In this type of application, each category often comes from a separate table. For example, Product categories come from the PRODUCTS table, while Region categories come from the REGIONS table. The data values for analysis, such as Units Sold and Cost, come from another table, SALES.

In the following diagram, the dimensions, all from different tables, are Customer, City, Region, Product, and Product Category. The facts, Units Sold and Cost, are in the Sales table. The multi-dimensional index, given any combination of dimensions, points directly to the relevant fact records in the fact table:



Each Fusion data source has an associated schema that describes every aspect of the data source including its indexes and joins. No run-time programming syntax is required to implement the features in a schema, making them transparent to the end user.

Fusion Data Sources

A data source is composed of files, classes, and fields. Fusion views a data source as a hierarchy of classes associated through parent-child relationships. A class is a collection of related fields, and a field is the smallest unit of data that an application can request. Each class is a child of the class directly above it in the hierarchy and the parent of all classes directly below it. A class can have multiple children or no children, but it can have only one parent. Only the class at the top of the hierarchy has no parent.

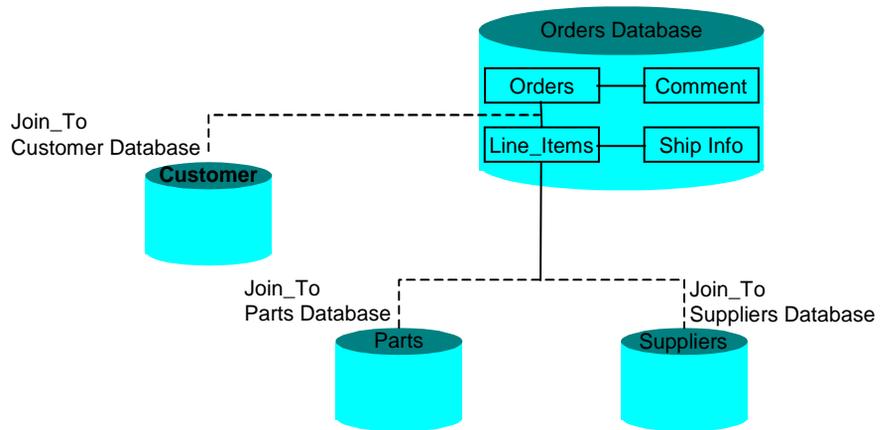
The data source is, logically, one entity; however, it can consist of many separate data sources. For example, one class can be stored separately from the rest of the data source records.

The Fusion Schema

Before accessing a data source with Fusion, you must describe its structure and relationships in a Fusion Schema. The Fusion Schema consists of:

- A Fusion Master File. Fusion views a data source as a hierarchy of classes. Each class is a collection of fields. The Fusion Master File describes the class hierarchy and the individual fields within it. The Fusion Master File can also manage:
 - Multi-dimensional indexes.
 - Remembered joins. Not only does Fusion automatically implement joins, it can also remember the locations of the joined records to optimize retrieval speed.
 - Derived data fields. Fusion can calculate new dimensions and measures and use them as if they were actual data source fields. The derived fields can also be MDI dimensions.
- A Fusion Access File. This is optional although highly recommended. The Fusion Access File provides comprehensive metadata management for all Fusion data sources. It coordinates all Fusion data sources involved in an application, and manages historical and partitioned data.

The following diagram illustrates a possible Fusion Schema structure:



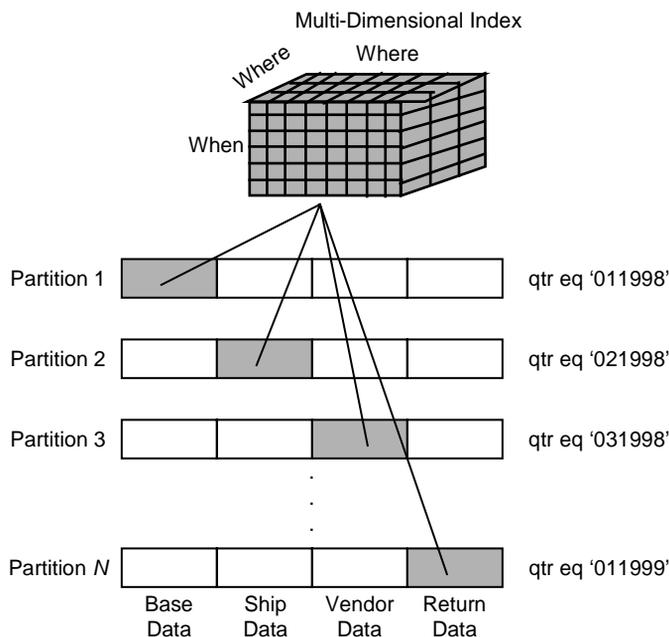
Appendix A, *Sample Schemas*, lists the Schema definition for this structure. For a detailed description of Fusion Master Files and Fusion Access Files, see Chapter 2, *Understanding Your Fusion Schema*.

High-Performance Features

With Fusion high-capacity data sources, you can accelerate retrieval by creating MDIs and intelligently separating data into partitions that optimize the use of system resources.

The Multi-Dimensional Index

A Fusion multi-dimensional index (MDI) knows where to find the relevant facts for any combination of its dimensions, across partitions. Therefore, a query that asks complex what-if questions based on multiple dimensions can use the MDI for fast access to the required data.



Most indexes locate information in the data source based on combining their dimensions in one inflexible order. As a result, only queries whose selection criteria involve all high-order dimensions can use the index (for example, clustered indexes in an RDBMS). In contrast, the Fusion MDI operates on any subset of its dimensions in any order.

The MDI is a single index that:

- Is aware of all dimensions.
- Is not order-dependent.
- Is smarter than a B-tree or bit-mapped index.

- Can selectively process only needed dimensions.
- Can process historical data transparently.

MDI dimensions are built directly into the Fusion Schema which makes them transparent to the end user and easy to maintain.

Intelligent Partitions

A partition is a separate data source that stores parts of a data source. Fusion supports two types of partitions, defined directly in the Fusion Schema:

- **Horizontal.** A horizontal partition separates the data source into files based on user-selected categories, such as year or region, providing a simple way to maintain historical data. The schema can even include a predicate that describes the values in each partition.

The separate partitions remain one logical data source and, using the information provided in the predicates, Fusion can selectively access only those partitions needed to satisfy each query. Updating the data source for a new period is a simple process of adding a new partition to the Schema; pre-existing partitions remain available at all times.

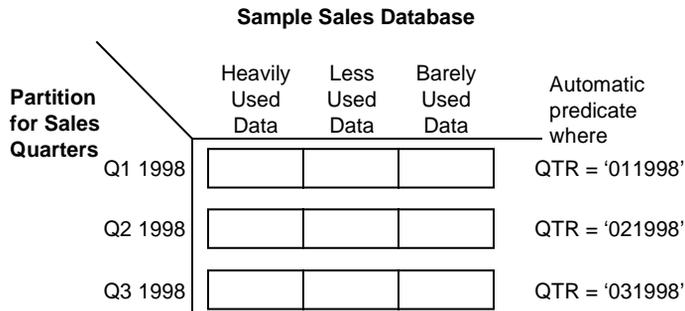
- **Vertical.** A vertical partition separates infrequently-accessed pieces of data source records from the rest of the data, minimizing retrieval overhead and memory usage.

See Chapter 3, *Using Fusion's High-Performance Features*, for instructions on choosing multi-dimensional index dimensions and creating horizontal and vertical partitions.

Example

Using Partitions

The following diagram depicts a data source that is organized into intelligent partitions:



Database Administration Tools

To use any data source management system effectively, you must be able to perform administrative tasks easily and quickly. With a stored procedure, you can perform administrative tasks, such as loading a Fusion data source and migrating from FOCUS to Fusion.

See Chapter 4, *Using a Multi-Dimensional Index*, for details of data source administration.

Metadata Management

Metadata means “data about data.” One example of metadata is a Fusion Schema. The FUSCAT Master File is a tool for querying your Fusion Schemas. With it, data source administrators and end users can determine which data sources are available and what information is available in a data source.

See Chapter 7, *The Fusion Catalog*, for details.

CHAPTER 2

Understanding Your Fusion Schema

Topics:

- The Fusion Schema
- Fusion Declarations
- The Fusion Master File
- The Fusion Access File

Your Fusion Schema governs the behavior of Fusion. It contains Master Files and Access Files that add functionality to your data sources. This topic explains how to create Master Files and Access Files.

Note: All sample Access Files in this chapter use the paths that are valid for Windows NT.

The Fusion Schema

A Fusion Schema describes a data source; that is, it describes all the participating files, classes, and fields and the relationships between them. The schema can also describe how to concatenate and retrieve the information from the data source.

For FOCUS users, you can build or edit a Fusion Schema with any text editor. See Chapter 4, *Database Administration Facilities*, for information about automatically creating a Fusion Schema when migrating other data source types to Fusion.

Reference

Fusion Schema Components

The Fusion Schema consists of:

- A Fusion Master File that describes the classes, fields, and relationships that constitute the data source. See *The Fusion Master File* on page 2-3.

The Fusion Master File can describe derived fields—fields that are not actually stored in the data source. Fusion includes an extensive syntax for writing expressions that describe how to calculate the value of a derived field. See Chapter 5, *Creating Derived Fields*.

- A Fusion Access File. This file is optional, but highly recommended, and describes how to concatenate, select, or join the various data files in the data source. See *The Fusion Access File* on page 2-31.

Fusion Declarations

A Fusion Master File and Fusion Access File are free-format text files composed of declarations. Each declaration consists of attributes and values assigned to them. Physical file name attributes can be in lower case; all other values must be in upper case.

Blank spaces separate declarations, attributes, commands, and values from each other. Use single quotation marks around descriptions or text strings that include blank spaces. Column alignment and line breaks have no significance; a new declaration or attribute can begin on the same line as the previous one and can continue onto as many lines as necessary.

Syntax

How to Create a Declaration

```
attribute_1 value_1 [attribute_2 value_2 ... attribute_n value_n]
```

where:

```
attribute_1...attribute_n
```

Are attributes that are part of the declaration.

```
value_1...value_n
```

Are acceptable values for their respective attributes.

All of the attributes that apply to a particular declaration must immediately follow each other in the schema. Most declarations must begin with a specific attribute that identifies the type of declaration; the order of other attributes within the declaration is irrelevant.

The Fusion Master File

A Fusion Master File is composed of declarations that describe the classes and fields in a data source, and the relationships between them. The Fusion Master File consists of the following declarations:

- One file declaration. Each Fusion Master File begins with a file declaration that assigns a logical name to the data source and identifies the type of data source described by the Schema.
Note: The file declaration can also identify a Fusion Access File and describe how to assign the appropriate century to dates stored with two-digit years.
- The REMARKS attribute can be used to provide a full description of the data source. This attribute is optional.
- One or more class declarations. Each class declaration assigns a name to the class, identifies its parent in the class hierarchy, specifies the number of key fields for the class, and optionally allocates the class data to a separate physical data source.
- One or more field declarations. Each field declaration assigns a name to the field and describes its data type and length.

A field declaration can also assign an alternate name for the field, a title that appears with the field in reports, a description of the field contents, and error message text for the field. A field declaration can also create a derived field, specify index information, indicate whether the field supports missing data, and describe how to assign the appropriate century to dates stored with two-digit years.

Reference

Guidelines for Using a Master File

The following guidelines govern the use of a Fusion Master File:

- Every request must include the source name of a Fusion Master File. Fusion then locates the Fusion Master File and uses it to access the data source. The default physical file for a data source has the same name as the Fusion Master File. The default is an extension of .fus, a file type of FUSION, or a data set extension of FUSION.
- The source of the Fusion Master File must have either the extension .mas or the file type MASTER, or it must be a member of the MASTER or EDAMFD data set depending on your operating environment.
- A Fusion Master File can contain comments, which are explanatory text that has no significance to Fusion. A comment declaration begins with a dash and asterisk (-*). Although a comment can begin at any point on a line, it always continues to the end of the line; Fusion does not recognize any syntax to the right of the comment. To continue a comment onto another line, you must repeat the comment declaration characters (-*) on the new line.

It is highly recommended that you use a Fusion Access File to have full control of your Fusion data source location and names.

Example

Sample Fusion Master File

The following example is a Fusion Master File describing the BUILDING data source:

```
FILENAME BUILDING FILETYPE FUSION
CLASS BUILDING KEYS S1
FIELD BUILDING_ID FORMAT A8 INDEX ON
FIELD ADDRESS FORMAT A48
FIELD STATE FORMAT A16
FIELD CITY FORMAT A16
FIELD ZIP FORMAT A10
```

File Attributes

File attributes perform such tasks as assigning a logical name to a file, identifying a data source type, and supplying descriptive information. For such tasks, use the following attributes in the file declaration:

Attribute	Synonym	Purpose
FILENAME	FILE	Assigns a logical name to the file.
FILETYPE	SUFFIX	Identifies the type of data source.
ACCESSFILE	ACCESS	Identifies the Fusion Access File associated with the Master File.
REMARKS	DESCRIPTION	Supplies descriptive text for documentation purposes and metadata management.
FDEFCENT	FDFC	Used with FYRTHRESH, it identifies the correct century for date values stored with a two-digit year.
FYRTHRESH	FYRT	Used with FDEFCENT, it identifies the lowest year that applies to the century indicated in FDEFCENT.
DATASET	DATA	Assigns a physical file to the data source.

All file declarations must begin with a FILENAME attribute and include a FILETYPE attribute.

Syntax

How to Create a File Attribute

```
{FILENAME|FILE} filename {FILETYPE|SUFFIX} FUSION
  {ACCESSFILE|ACCESS} accessname
  {DATASET|DATA} dataname
  {FDEFCENT|FDFC} {cc|19} {FYRTHRESH|FYRT} {nn|00}
  {REMARKS|DESCRIPTION} desc.string
```

where:

filename

Is the logical file name of the Fusion Master File.

accessname

Is the logical file name of the Fusion Access File.

dataname

Is the fully qualified physical file name of a single file that serves as a data repository for the data source.

nn

Is the two-digit threshold year for determining the century. The default value is 00.

desc.string

Is a descriptive string of text. If it includes spaces, the string must be enclosed in single quotation marks.

cc

Is the two-digit century to be used. The default value is 19.

Note: Either the ACCESSFILE or DATASET attribute must be present, but these attributes cannot be used together.

Assigning a Logical Name to a File: FILENAME

Assign a logical name to a file with the FILENAME attribute. This attribute must be the first attribute in a file declaration.

Syntax

How to Assign a Logical Name to a File

`{FILE|FILENAME} filename`

where:

filename

Is a one- to eight-character logical name for the data source file. The name can consist of letters, digits, and underscore.

Identifying the Type of Data Source: FILETYPE

The FILETYPE attribute identifies the type of data source described by the Schema.

Syntax

How to Identify the Data Source Type

`{FILETYPE|SUFFIX} FUSION`

where:

`FUSION`

Indicates that the data is stored in a Fusion data source.

Example

Identifying the Data Source Type

The following

```
FILENAME CAR FILETYPE FUSION
```

indicates that the data is stored in a Fusion data source.

Identifying a Fusion Access File: ACCESSFILE

Identify a Fusion Access File with the ACCESSFILE attribute. The Fusion Access File describes how to find and concatenate the appropriate data sources for a request against the Fusion Master File.

Syntax

How to Identify a Fusion Access File

```
{ACCESS|ACCESSFILE} accessname
```

where:

accessname

Is the logical name of the Fusion Access File to use when retrieving data with the Fusion Master File. The Fusion Access File must have an extension of .acx or a file type of ACCESS, or it must be a member of the ACCESS or EDAAFD data set depending on your operating environment.

Example

Identifying a Fusion Access File

A Fusion Access File can manage all partitions and locations for one Fusion Master File or for multiple Fusion Master Files. For example, Fusion Master Files ONE and TWO are described in the same Access File:

```
FILENAME ONE FILETYPE FUSION ACCESSFILE SALESAPP
FILENAME TWO FILETYPE FUSION ACCESSFILE SALESAPP
```

Identifying a Repository for a Data Source: DATASET

You can identify a single physical file that serves as the repository for all data in a data source with the DATASET attribute.

Syntax

How to Identify a Repository for a Data Source

```
{DATASET|DATA} dataname
```

where:

dataname

Is the fully qualified physical file name of a single file that serves as a data repository for the data source.

Note: A DATASET attribute cannot coexist with an ACCESSFILE attribute.

Supplying Descriptive Text: REMARKS

Supplies descriptive text for documentation purposes and metadata management.

Syntax

How to Supply Descriptive Text

```
{REMARKS|DESCRIPTION} desc.string
```

desc.string

Is a descriptive string of text. If it includes spaces, the string must be enclosed in single quotation marks.

Defining the Correct Century for a File: FDEFCENT AND FYRTHRESH

Many existing business applications use two digits to designate a year, instead of four digits. When they receive a value for a year, such as 00, they typically interpret it as 1900, assuming that the first two digits are 19, for the twentieth century. These applications require a way to handle dates when the century changes (for example, from the twentieth to the twenty-first), or when they need to perform comparisons or arithmetic on dates that span more than one century.

The FDEFCENT and FYRTHRESH attributes provide a file-wide mechanism for controlling the century value of date fields defined with two-digit years. Equivalent attributes (DEFCENT, YRTHRESH) exist at the field level (see *Field Attributes* on page 2-15). The field level attributes override FDEFCENT and FYRTHRESH for those fields on which they are defined.

FDEFCENT and FYRTHRESH together define a 100 year window. FDEFCENT supplies a two-digit default century. FYRTHRESH indicates the lowest year to which this century value applies, that is the century threshold. For example, if FDEFCENT is 19 and FYRTHRESH is 80, year values from 80 through 99 are interpreted as 1980 through 1999, and year values from 00 through 79 are interpreted as 2000 through 2079.

If you are satisfied with the FDEFCENT and FYRTHRESH default values, you can omit one or both of these attributes from the Fusion Master File.

Syntax

How to Define the Default Century

```
{FDEFCENT|FDFC} {cc|19}
```

where:

cc

Is a two-digit century. The default is 19.

Syntax

How to Define Where a New Century Begins

```
{FYRTHRESH|FYRT} {nn|00}
```

where:

nn

Is a two-digit number indicating the lowest year to which FDEFCENT applies. The default is zero.

Example

Defining the Correct Century

The following example

```
FILENAME EMPLOYEE FILETYPE FUSION FDEFCENT 20 FYRTHRESH 66
```

assigns 2000 as the default century, and 66 as the threshold year.

Class Attributes

A class is a collection of related fields, and a field is the smallest unit of data that an application can request. A class declaration describes the format of the records in the class. Each actual record is called an instance of the class.

Class declarations allow you to perform such tasks as assigning a name to a class, describing a relationship between parent and child, and separating certain fields in the class from the rest of the fields. Use the following attributes in your class declaration to perform such tasks:

Attribute	Synonyms	Purpose
CLASSNAME	CLASS	Assigns a name to the class.
PART_OF	PARENT	Describes a one-to-many relationship between a parent and its child. The parent class can have multiple instances of a class that is PART_OF the parent.
SUBCLASS_OF		Describes a one-to-one relationship between a parent and a child. The parent class can have at most one instance of a class that is a SUBCLASS_OF the parent.
CLASSKEYS	CLASSKEY KEYS KEY	Identifies the number of key fields in the class, and the sort order for instances of the class. Only used for the root class or a class with the PART_OF attribute.
TARGET_OF		Names an MDI that provides access to the class.
LOCATION		Physically separates the class data from the rest of the data source. Assigns a logical name to the file that contains the data for the class. The Fusion

Attribute	Synonyms	Purpose
		Access File supplies the physical file name.
SECTION		Separates certain fields in the class from the rest of the fields (vertical partitioning). Use with the LOCATION attribute to specify the logical file name for the section data. The Fusion Access File supplies the physical file name.

All class declarations must begin with a CLASSNAME attribute. In addition, each child class must identify its parent and relationship to the parent.

Syntax

How to Identify Class Attributes

```
{CLASSNAME|CLASS} classname SUBCLASS_OF parentname
{PART_OF|PARENT} parentname
classtype {SHn|Sn|S0}
[TARGET_OF mdiname1 [TARGET_OF mdiname2]...]
[LOCATION locationname]
```

where:

classname

Is a one- to eight-character name consisting of letters, digits, and underscores.

parentname

Is the name of the parent class, a class previously described in the Fusion Master File. This syntax can be omitted for the top class, which has no parent.

classtype

Is one of the following synonyms: CLASSKEY, CLASSKEYS, KEY, or KEYS.

S*n*

Sorts the data in ascending order. *n* is the number of fields that participate in the class key. The first *n* fields described immediately following the class declaration are included.

SH*n*

Sorts the data in descending order. *n* is the number of fields that participate in the class key. The first *n* fields described immediately following the class declaration are included.

S0

Stores data as unkeyed and unsorted. It is stored in the physical order in which it is input.

mdiname1,mdiname2

Are logical names of multi-dimensional indexes.

locationname

Is a one- to eight-character name consisting of letters, digits, and underscore characters.

Assigning a Name to a Class: CLASSNAME

Assign a name to a class with the CLASSNAME attribute. It must be the first attribute in any class declaration.

Syntax

How to Assign a Name to a Class

```
{CLASSNAME | CLASS} classname
```

where:

classname

Is a one- to eight-character name consisting of letters, digits, and underscores.

Indicating the Sort Sequence and Number of Fields in the Key: CLASSKEYS

A key is a field or group of fields that identify each instance of a class. A key is called unique when no two class instances have the same key value; in a Fusion data source, the class key is always unique.

The CLASSKEY attribute indicates the sort sequence of the class instances and the number of fields that constitute the key. The class instances can be stored in ascending or descending key sequence.

Immediately following the class declaration, the key fields must be listed prior to any non-key fields in the class. You can describe the non-key fields in any order, after all key fields.

Syntax

How to Indicate the Sort Sequence and Number of Fields in a Key

```
classtype {Sn|SHn|S0}
```

where:

classtype

Is one of the following synonyms: CLASSKEY, CLASSKEYS, KEY, or KEYS.

Sn

Sorts the data in ascending order. *n* is the number of fields that participate in the class key. The first *n* fields described immediately following the class declaration are included.

SHn

Sorts the data in descending order. *n* is the number of fields that participate in the class key. The first *n* fields described immediately following the class declaration are included.

S0

Stores data as unkeyed and unsorted. It is stored in the physical order in which it is input.

Example

Identifying a Sort Sequence

In the following example, the key consists of the first field described following the class declaration: EMPLOYEE_NUMBER. The instances are stored in ascending EMPLOYEE_NUMBER order:

```
CLASS EMPLOYEE CLASSKEYS S1
  FIELD EMPLOYEE_NUMBER
  FIELD LAST_NAME
  FIELD FIRST_NAME
```

Identifying the Target Class: TARGET_OF

A multi-dimensional index uses dimension values from one or more classes to find facts (data values for analysis) that may exist in an entirely separate class or data source. For more information on multi-dimensional index dimensions and facts, see Chapter 4, *Database Administrator Facilities*.

The target class for the index is the class that contains the facts. You must identify the target class with the TARGET_OF attribute. If the target data is distributed among several classes, the target class should be the class that contains most of the analysis data.

Syntax

How to Identify the Target Class

```
TARGET_OF mdiname_1 [TARGET_OF mdiname_2...]
```

where:

mdiname_1, mdiname_2

Are logical names of multi-dimensional indexes.

Note: You can determine the logical name for an index from the LOCATION attribute in the relevant FIELD declaration.

Assigning a Logical Name to a File: LOCATION

Fusion provides several techniques for optimizing performance and managing data source size. One important technique is the ability to store certain classes in separate physical files. The LOCATION attribute assigns a logical name to a file.

Syntax

How to Assign a Logical Name to a File

LOCATION *locationname*

where:

locationname

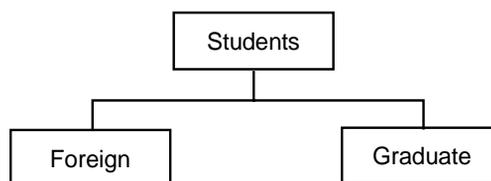
Is a one- to eight-character name consisting of letters, digits, and underscores.

Describing Class Relationships: SUBCLASS_OF and PART_OF

The Fusion Schema describes classes as related hierarchically. Each class, except the top class, has a parent; it may also have children. Two types of relationships exist:

- **SUBCLASS_OF:** The parent and child participate in a one-to-one relationship; a parent can not have more than one instance of this type of child.

A subclass is a type of its parent; that is, each subclass defines a separate record type with its own fields. The child is a more specific example of its parent. The following diagram illustrates the subclass relationship:



Here, the Students class is the parent. The children are more specific types of the student, Foreign and Graduate. A subclass cannot have a CLASSKEY attribute; its keys and sort sequence are identical to those of its parent. However, a subclass can have children. If the data source contains values for a parent instance, but does not contain data for a corresponding subclass instance, the subclass is considered missing. In this case, a report that displays data from the parent instance displays the missing data symbol (.) for any printed subclass fields.

- **PART_OF:** The parent and child participate in a one-to-many relationship; a parent can have multiple instances of this type of child.

A part is a component of its parent. For example, a city is a part of a state, and an order item is part of an invoice.

Syntax

How to Identify a Parent-Child Relationship in a Class Declaration

```
{SUBCLASS_OF|PART_OF} parentname
```

where:

parentname

Is the name of a parent class previously described in the Fusion Master File. The top class has no parent and can omit this syntax.

Example

Identifying a Parent-Child Relationship in a Class Declaration

The following

```
CLASS GRADUATE SUBCLASS_OF STUDENT
```

identifies STUDENT as the parent, and GRADUATE as the subclass (child) of STUDENT.

Vertical Partitioning: SECTION

You can separate groups of fields from a class into sections. This avoids accessing and using up memory for infrequently-used fields, saves space in the main file, and organizes data across devices for system optimization. If you use a separate physical file to store the section data, the section is called a vertical partition.

You use the SECTION attribute to create vertical partitions. All fields described following the section declaration belong to the section; the section ends when a new section or class declaration starts. This cannot be the initial declaration for the class.

Syntax

How to Create a Vertical Partition

```
SECTION sectionname [LOCATION locationname]
```

where:

sectionname

Is a one- to eight-character name consisting of letters, digits, and underscores.

locationname

Is the logical name for the file that contains the section data.

Example

Using Vertical Partitions

The COMSEG section of the ORDERS Fusion Master File (found in Appendix A, *Sample Schemas*) consists of a comment field:

```
CLASS S_ORDERS
  FIELD ORDERKEY   FORMAT I9   INDEX ON
  .
  .
  .
SECTION COMSEG
  FIELD COMMENT    FORMAT A49
```

A section has a one-to-one relationship with the rest of the class; therefore, any report that displays values for a class instance will also display values for all fields in the section. If the data source does not contain data for the section instance, the report displays default values: zeros, blanks, or missing data symbols, depending on the individual field formats.

Field Attributes

Field attributes allow you to perform such tasks as assigning a name to a field, describing the format of a field in an external data source, and setting a default century. To perform such tasks, use the following attributes in your field declaration:

Attribute	Synonym	Purpose
FIELDNAME	FIELD	Assigns a name to the field.
ALIAS		Assigns an alternate name to a field.
USAGE	FORMAT	Describes how to display the field's values.
ACTUAL		Describes the format of the field in the external data source.
DEFCENT	DFC	Used with YRTHRESH. Identifies the correct century for date values stored with a two-digit year.
YRTHRESH	YRT	Used with DEFCENT. When date values are stored with a two-digit year, YRTHRESH identifies the lowest year that applies to the

Attribute	Synonym	Purpose
		century indicated in DEFCENT.
TITLE		Supplies a heading to appear with the field in reports.
MISSING		Indicates whether the field supports missing data.
ACCEPTVALUES	ACCEPT	Defines valid data values for the field.
HELPMESSAGE	HELPMMSGTEXT	Supplies text to display when there is an error using or entering data for the field.
DESCRIPTION	DEFINITION	Supplies descriptive text for documentation purposes and metadata management.
GROUPNAME	GROUP	Assigns one name to multiple fields.
INDEX		Indexes the field.
LOCATION		Used with INDEX. Assigns to the index a location for the field that differs from the storage location of the data.
MAXVALUES	MAXVAL	Used with INDEX. Indicates the number of distinct values for the index dimension or field.
WITHIN		Used with multi-dimensional indexes (MDI). Indicates a hierarchical relationship among the dimensions.
WITH		Used with DEFINE. Places a derived field in a specific class.
RJOIN_TO	RJOINTO	Creates a unique remembered join; stores the locations of the target fields for reuse.
JOIN_TO	JOIN	Creates a unique join, and locates target fields at run time.
JOIN_TO ALL	JOINALL	Creates a multiple join, and locates target fields at run time.
ONLY		Limits a join to one target class rather than the entire target data source. You can add additional related classes with LINK_TO.
LINK_TO	LINK	Adds ancestors or descendants of the target class to a join structure.

All field declarations must begin with a FIELDNAME attribute and include a USAGE attribute.

Syntax

How to Create a Field Attribute

```
{FIELDNAME|FIELD} fieldname ALIAS aliasname
{FORMAT|USAGE} format_type ACTUAL format_type
{DEFCENT|DFC} {cc|19} {YRTHRESH|YRT} {nn|00}
{TITLE title_string} {MAXVALUES|MAXVAL} n

MISSING {ON|OFF}
{ACCEPTVALUES|ACCEPT} value1 [ [OR] value2 ...][value1 TO value2]
FIND(indexed_fieldname IN filename)

{HELPMESSAGE|HELPMMSGTEXT} help_string
{DESCRIPTION|DEFINITION} desc_string
INDEX {ON|OFF} LOCATION indexname
INDEX MD_EXTERNAL LOCATION externalname {MAXVALUES|MAXVAL} n2 WITHIN
dimensionname

[[R] JOIN_TO [ALL]]|[R] JOIN [ALL]] mastername. [classname.] targetfield
ONLY
{LINK|LINK_TO|AND} [mastername.] classname
```

where:

fieldname

Is a 1- to 66-character name consisting of letters, digits, and underscores.

aliasname

Is any 1- to 66-character name consisting of letters, digits, and underscores.

format_type

Is a valid format type for the field.

cc

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

nn

Is a two-digit year. The default value is 00.

title_string

Is the text that will appear as a title. The string must be enclosed in single quotation marks if spaces appear in the title.

n

Is the maximum number of distinct values a field can have. This value must be a positive integer.

value1,value2

Define acceptable values.

indexed_fieldname

Is an indexed field containing the acceptable values.

filename

Is the Fusion or FOCUS data source that contains the field with the acceptable values.

help string

Is one line of text, up to 78 characters, enclosed in single quotation marks.

desc string

Is descriptive text about the field. This string can contain up to 43 characters of text, and must be enclosed in single quotation marks.

indexname

Is the logical name of the index.

externalname

Is the logical name of the external index.

n2

Is the maximum number of values for the dimension for each one value of the WITHIN index field. This value applies only to a multi-dimensional index.

dimensionname

Is the name of a dimension that is participating in this MDI.

mastername

Is the fully qualified physical file name of the Fusion Master File.

classname

Is any one- to eight-character name consisting of letters, digits, and underscores.

targetfield

Is the name of the join field in the target class. This field must be indexed; it can be part of an MDI or have a standard or external B-tree index.

Assigning a Name to a Field: FIELDNAME

The FIELDNAME attribute assigns a name to the field. It must be the first attribute in each field declaration. The field name is the default heading used with the field in reports; therefore, it should describe the contents of the field.

Within a class, multiple fields cannot have both the same field name and alias. If duplicate field names exist in *separate* classes, requests can indicate which field to access by qualifying the field name or alias with the appropriate file name and/or classname: *filename.fieldname*, *filename.classname.fieldname*, or *classname.fieldname*. When a request uses a qualified field name, the entire name, including the qualifiers and qualification characters (the periods that follow the qualifiers) must not exceed 66 characters.

Syntax

How to Assign a Name to a Field

```
{FIELDNAME|FIELD} fieldname
```

where:

fieldname

Is a 1- to 66-character name consisting of letters, digits, and underscores.

Assigning an Alternate Name to a Field: ALIAS

Use the ALIAS attribute to provide a short, easy name when the field name is long and cumbersome, or you can use it to distinguish between fields that have the same field name.

Syntax

How to Assign an Alternate Name to a Field

```
ALIAS aliasname
```

where:

aliasname

Is any 1- to 66-character name consisting of letters, digits, and underscores.

Describing a Field's Data Type and the Display of the Field Value: USAGE

The USAGE attribute describes the field's data type and how to display the field value. The display format must be one of the following:

- **Numeric.** There are four types of numeric formats: integer, single-precision floating-point, double-precision floating-point, and packed decimal.
- **Alphanumeric.**
- **Date.** You can use a date format to define date components such as year, quarter, month, day, and day of week. You can also compare dates, do arithmetic with dates, and automatically validate dates in transactions. An alternative to using a date format is to use alphanumeric, integer, or packed-decimal fields with date display options.
- **Text.**

Syntax

How to Describe a Field's Data Type and the Display of the Field Value

```
{USAGE|FORMAT} datatype length [option]
```

where:

datatype

Is the data type. Valid values are A (alphanumeric), D (floating-point double-precision), F (floating-point single-precision), I (integer), P (packed decimal), and TX (text), as well as date data types composed of a valid combination of the components D, W, M, Q, and Y. For complete information on data types, see Appendix C, *Using Data Types and Display Options*.

length

Is a length specification. Different data types have different length specifications; see Appendix C, *Using Data Types and Display Options*.

option

Are one or more display options. Different data types offer different display options; see Appendix C, *Using Data Types and Display Options*.

The complete USAGE value cannot exceed eight characters. For most data sources, you can change the type and length specifications only to other types and lengths valid for that field's ACTUAL attribute. You can change display options at any time.

Defining the Format of a Field: ACTUAL

The ACTUAL attribute defines the format of the field as seen in the external data source. It is not needed for Fusion data sources.

Syntax

How to Define the Format of a Field

`ACTUAL datatype length [option]`

where:

datatype

Is the data type. Valid values are A (alphanumeric), D (floating-point double-precision), F (floating-point single-precision), I (integer), P (packed decimal), and TX (text), as well as date data types composed of a valid combination of the components D, W, M, Q, and Y. For more information see Appendix C, *Using Data Types and Display Options*.

length

Is a length specification. Different data types have different length specifications; see Appendix C, *Using Data Types and Display Options* for more information.

option

Are one or more display options. Different data types offer different display options; See Appendix C, *Using Data Types and Display Options* for more information.

Defining the Correct Century for a Field: DEFCENT AND YRTHRESH

The DEFCENT and YRTHRESH attributes provide a mechanism for controlling the century value in a particular field. Equivalent attributes (FDEFCENT, FYRTHRESH) exist at the file level. The field level attributes override the file level attributes for those fields on which they are defined.

DEFCENT and YRTHRESH together define a range of 100 years. DEFCENT specifies a two-digit century number. YRTHRESH indicates the lowest year to which this century applies. For example, if DEFCENT is 19 and YRTHRESH is 80, year values from 80 through 99 are interpreted as 1980 through 1999, and year values from 00 through 79 are interpreted as 2000 through 2079.

If you are satisfied with the DEFCENT and YRTHRESH default values, you can omit one or both of these attributes from the Fusion Master File.

In the FIELD declaration, the DEFCENT and YRTHRESH attributes must follow the USAGE attribute.

You can also define DEFCENT and YRTHRESH values in derived fields. For information, see Chapter 5, *Creating Derived Fields*.

Syntax

How to Define the Default Century

```
{DEFCENT|DFC} {cc|19}
```

where:

cc

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

How to Define the End of the Default Century

```
{YRTHRESH|YRT} {nn|00}
```

where:

nn

Is a two-digit number indicating the lowest year that applies to DEFCENT. The default value is 00.

Example

Defining the Correct Century

The following example

```
FIELDNAME HIRE_DATE ALIAS HDT USAGE I6YMD DEFCENT 19 YRTHRESH 75
```

defines the default century as the twentieth, and the year threshold as 75.

Supplying a Column Title for a Field on a Report: TITLE

The TITLE attribute supplies a column title for a field on a report. If omitted, the field name is the default column title.

Syntax

How to Define a Column Title

```
TITLE text[,text]
```

where:

text

Is up to 64 characters of text, enclosed in single quotation marks.

,

Indicates a line break. The title can be broken up onto as many as five lines.

Example

Defining a Column Title

The following

```
TITLE 'Driver,License No.'
```

displays as:

```
Driver  
License No.
```

Limiting the Number of Unique Values for a Field: MAXVALUES

The MAXVALUES attribute limits the number of unique values for this field. This is useful in deciding the space taken by this field when it is used as a dimension in an MDI.

Syntax

How to Limit the Number of Unique Values

```
MAXVALUES n
```

where:

n

Is the maximum number of distinct values the field could have. This value must be a positive integer.

Missing Data Support: MISSING

Data for a field can be unknown or irrelevant. For example, a payroll file may not have a raise amount for every employee.

Missing data is treated as a zero or blank. However, it may not always be appropriate to treat a missing value as if it were a zero or blank. For example, adding extra zeros into a sum or average may produce a misleading result. Missing data support can be used in this case by setting the MISSING attribute. Fusion can distinguish missing values from zero or blank values, and omit them from calculations, when the field declaration indicates missing value support.

Syntax

How to Set Missing Data Support

`MISSING {ON|OFF}`

where:

`ON`

Supports missing values.

`OFF`

Stores a blank in an alphanumeric field, or a zero in a numeric field when data is missing. OFF is the default.

Identifying Acceptable Values for a Field: ACCEPTVALUES

The ACCEPTVALUES attribute identifies a list, range, or file of acceptable values for the field.

Syntax

How to Identify Acceptable Values for a Field

`{ACCEPTVALUES|ACCEPT} [value_1 [[OR] value_2] ...][value_1 TO value_2]`
`[FIND(fieldname IN filename)]`

where:

`value_1, value_2`

Defines a list or range of acceptable values. Any values containing embedded blanks must be enclosed in single quotation marks.

`FIND`

Verifies incoming data against a file that contains the acceptable values.

`fieldname`

Is an indexed field that contains the acceptable values.

`filename`

Is the Fusion or FOCUS data source that contains the field with the acceptable values.

Example

Identifying Acceptable Values

The following example illustrates settings for acceptable values:

```
ACCEPT MR OR MRS OR MS OR MISS OR DR
ACCEPT MR MRS MS MISS DR
ACCEPT 15 TO 30
ACCEPT FIND(CARNAME IN CARFILE)
```

The following example encloses BLUE GREEN in single quotation marks because there is an embedded blank:

```
ACCEPT BLUE OR GREEN OR 'BLUE GREEN'
```

Supplying Text to Display When a Value Fails a Validation Test: HELPMESSAGE

The HELPMESSAGE attribute supplies text that displays when an incoming field value fails an ACCEPT validation test or is entered with the incorrect format.

Syntax

How to Supply Text to Display After a Failed Validation Test

```
{HELMMESSAGE|HELPMMSGTEXT} help_string
```

where:

help_string

Is one line of text enclosed in single quotation marks. The string can contain a maximum of 78 characters.

Providing a Field Description: DESCRIPTION

The DESCRIPTION attribute provides a description of a field for documentation purposes and use by application tools, such as the Fusion Catalog (FUSCAT), that allow users to query the schema. It is important for the description to be as meaningful and complete as possible.

Syntax

How to Provide a Field Description

```
{DESCRIPTION|DEFINITION} desc_string
```

where:

desc_string

Is up to 43 characters of text, enclosed in single quotation marks.

Setting Index Attributes: INDEX

The INDEX attribute indexes a field. When a field has an index, Fusion can find specific field values directly without an exhaustive search of the data source. Two types of indexes are available with Fusion: a standard index and a multi-dimensional index:

- A standard (B-tree) index operates on one field in the data source. The index is maintained as part of the data source even if it is stored as a separate data source with its own LOCATION attribute.
- A multi-dimensional index (MDI) operates on multiple fields. Each field that participates in the multi-dimensional index is called a dimension.

An MDI is external to the data source. That is, it exists as a separate data source with its own LOCATION attribute. It requires rebuilding when the data source changes. You can use the WITHIN and MAXVALUES attributes to describe the MDI.

- Use the WITHIN attribute to indicate that one dimension is a subset of another dimension in an MDI (for example, City can be a subset of State).
- The MAXVALUES attribute has varying significance depending on its placement. The two possibilities are:
 - **A field parameter.** The MAXVALUES here indicates that the field has a certain number of distinct values. It is like any other field attribute, for example, FORMAT or ALIAS.

```
FIELD DEPARTMENT FORMAT A20 MAXVALUES 20
```

- **A WITHIN modifier.** A MAXVALUES attribute in an optional MD_EXTERNAL definition modifies the WITHIN structure so that the provided value represents the number of distinct values that exist in the WITHIN structure.

```
FIELD DEPARTMENT FORMAT A20 MAXVALUES 20
      MAXVALUES 10 INDEX MD_EXTERNAL LOCATION MDI1 WITHIN DIVISION
```

Using the MAXVALUES attribute to indicate this relationship enables Fusion to minimize the space allotted for the index.

If your data source has multiple MDIs, the AUTOINDEX feature chooses the best index for each query. See Chapter 3, *Using Fusion's High-Performance Features*.

Syntax

How to Set the Index Attribute

```
INDEX {ON|OFF|MD_EXTERNAL} [LOCATION indexname]
```

where:

ON

Creates a standard B-tree index on the field.

`OFF`

Does not index the field. This value is the default.

`MD_EXTERNAL`

Creates a dimension in the MDI referred to by *indexname*.

`LOCATION`

Is required when a field is indexed.

indexname

Is the logical name for the index. This is a one to twelve character name.

Note:

- All dimensions in a multi-dimensional index must be on the same path in the class hierarchy. If you need indexed access to separate paths, define separate MDIs for each path, and make their common parent class the target class for each index.
- The target class for an MDI must be in the host file when the index is defined across joined files.

Syntax

How to Identify Additional Attributes for a Multi-Dimensional Index

`[MAXVALUES n] WITHIN dimensionname`

where:

n

Is the maximum number of values for the dimension for each one value of the WITHIN index field. This value applies only to a multi-dimensional index.

dimensionname

Is the name of another dimension that is participating in this MDI.

Example

Identifying Additional Attributes for a Multi-Dimensional Index

The following uses the MAXVALUES attribute to specify that each state can have up to 100 cities:

```
FIELD CITY INDEX MD_EXTERNAL LOCATION MD1  
MAXVALUES 100 WITHIN STATE
```

All external indexes require maintenance when the data source is updated or when a new horizontal partition is added. For information about maintaining an index, see Chapter 4, *Database Administration Facilities*.

Describing a Group of Fields

A group declaration can assign one name to multiple fields. The required attributes for a group declaration are `GROUPNAME` and `USAGE`. You can include `INDEX` attributes in order to access the group with an index. The group may also be the key for the class.

The field declarations for the subordinate fields that participate in the group must immediately follow the group declaration.

Syntax

How to Describe a Group of Fields

```
{GROUPNAME|GROUP} groupname USAGE format
INDEX {ON|OFF} [LOCATION indexname ]
INDEX MD_EXTERNAL LOCATION externalname_1 {MAXVALUES|MAXVAL}n
WITHIN dimensionname
[INDEX MD_EXTERNAL LOCATION externalname2 ...] ...
```

where:

groupname

Is the group name. This can be any name that complies with Fusion field naming conventions.

format

Is the USAGE format for the group. The format must be alphanumeric (A), and its length must count all subordinate fields.

- Subordinate single-precision floating-point fields are counted as length 4, all decimal fields as length 8, and all integer fields as length 4.
- For alphanumeric fields, the sum of the lengths of subordinate fields must equal the group USAGE length.
- For packed decimal fields, the length depends on the USAGE length of the packed decimal subordinate field. If the subordinate fields add up to 15 or less, the group USAGE is 8. If the subordinate fields add up to between 16 and 33, the group USAGE is 16.

ON

Indexes the group of fields.

OFF

Does not index the group of fields.

indexname

Is the logical name for the index.

externalname_1

Is the logical name of the MDI.

n

Is the maximum number of values for the dimension for each one value of the WITHIN index field. This value applies only to a multi-dimensional index.

dimensionname

Is the name of another dimension that is participating in the MDI.

externalname2

Is the logical name of the MDI.

Example

Describing Group Names

The following example describes a group of fields:

```
GROUP    PHONE_NUMBER  USAGE A9 INDEX ON
FIELD   AREACODE      USAGE A3
FIELD   NUMBER        USAGE A6
```

Joining Classes: JOIN_TO, JOIN_TO ALL, and LINK_TO

A join is a method used to report from two separate data sources that have corresponding data types and values in one or more fields. It identifies matching instances from two files based on their common fields. The JOIN_TO, JOIN_TO ALL, and LINK_TO attributes are used to establish a join.

A join results in a hierarchy in which the source class is the parent and the target class is the child. The source and target classes can be related in the following ways:

- A *unique join* describes a one-to-one relationship between the source and target classes. That is, any instance of the source class can have only one matching instance in the target class. If multiple matching instances exist in the target file, Fusion retrieves only one.
- A *non-unique join* describes a one-to-many relationship between the source class and the target class. That is, any instance of the source class can have multiple matching instances in the target class.

A join can be implemented in two ways:

- It can store the locations of all matching target instances in the source data source. This type of join is called a remembered or static join. It requires additional space in the source data source. The locations are maintained automatically.
- It can find the matching target instances dynamically at run time.

Once the join is implemented, when Fusion retrieves a class instance from the first (source) file, it also retrieves matching data from the second (target) file.

Syntax

How to Join Classes

```
FIELD fieldname [R]{JOIN_TO|JOIN_TO ALL|JOIN ALL} mastername.classname
[ONLY]
{LINK|LINK_TO|AND} mastername.classname [AS alias]
```

where:

fieldname

Is the name of the field from the source data base included in the join.

R

Is a remembered join. If this value is not specified, the joins are formed dynamically at run time.

JOIN_TO

Creates a unique join. Each instance of the source class has only one matching instance in the target field.

JOIN_ALL, JOIN_TO ALL

Creates a non-unique join; Fusion retrieves all matching instances in the target class.

Note: RJOIN_TO ALL behaves exactly like SEGTYPE=KM in FOCUS.

mastername

Is the logical name of the target data source.

classname

Is the name of the target class to be included in the join.

ONLY

Limits the join to the specified target class.

LINK, LINK_TO

Adds a child or parent of the target class to the join. The parent or child to be included must follow a direct path up or down from the target class without skipping any levels.

AS *alias*

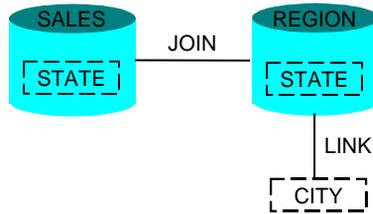
Assigns an alias name to the target class in a join. You can then prefix the target field names with the AS name in a request. For example, if the AS name is J1 and the field name is EMPID, the field name becomes J1EMPID in a request.

You need an AS name any time you join to the same target class more than once in the same Fusion Master File. The distinct field names produced by the AS name enables Fusion to determine which join to use in order to access the target field.

Examples

Joining Classes

The following diagram illustrates a join between the SALES data source and the REGION data source:



The following JOIN commands, issued in the SALES Fusion Master File, illustrate the various ways the two data sources can be joined.

- The field declaration

```
FIELD STATE JOIN_TO REGION.STATE
```

includes all three classes in the join.
- The field declaration

```
FIELD STATE JOIN_TO REGION.STATE ONLY
```

omits CITY from the join.
- The field declaration

```
FIELD STATE JOIN_TO REGION.STATE  
LINK_TO REGION.CITY
```

adds CITY back into the join.

Dynamically Joining Classes

The following declaration dynamically joins the EMPLOYID field to the EMPLOYCODE field in the PEOPLE data source. It retrieves all matching instances:

```
FIELDNAME EMPLOYID FORMAT I5 JOIN_TO ALL PEOPLE.EMPLOYCODE
```

For information on joining data sources with a multi-dimensional index, see *Describing Joined Files* on page 2-40.

The Fusion Access File

The Fusion Access File provides comprehensive metadata management for all Fusion files. 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 Fusion Access File describes how to locate, concatenate, join, and select the appropriate physical data sources for retrieval requests against one or more Fusion data sources. A Fusion Access File is optional in retrieval requests against non-partitioned data sources with no location files. It plays no part in data maintenance requests, except when rebuilding an MDI that is larger than two gigabytes.

In a request to Fusion, Fusion reads the supplied Fusion Master File and uses the schema declarations in it to access the data source. If the Fusion Master File includes an ACCESSFILE attribute, Fusion reads the named Fusion Access File and uses it to locate the correct data sources. A Fusion Access File can be pointed to by one Fusion Master File or several Master Files. This makes it possible to create one Fusion Access File that manages data source access for an entire application.

A Fusion Access File is necessary to take advantage of the following data source features:

- **Horizontal partitioning.** A data source can consist of several separate files, called partitions, each of which contains the data source records for a specific element such as time period or region. The Fusion Access File describes how to concatenate the separate data sources, and can also provide a logical predicate that describes the specific data values in each partition. Fusion uses this information to optimize data source access by retrieving only those partitions whose values are consistent with the selection criteria in the request.
- **Joins.** If joined files are partitioned, the Fusion Access File describes how to concatenate the separate data sources in the join.
- **Partitioned MDI for large MDIs.**

Example

Sample Fusion Access File

The following is a sample Fusion Access File:

```

MASTERNAME filename

-* The following is for the first horizontal partition.

    DATANAME horizontal partition 1 name
        [WHERE expression ;]
[LOCATION class location
    DATANAME class filename]
[INDEXLOCATION index location
    DATANAME index filename]

-* The following is for the second horizontal partition.

    DATANAME horizontal partition 2 name
        [WHERE expression ;]
[LOCATION class location
    DATANAME class filename]
[INDEXLOCATION index location
    DATANAME index filename]

-* The following is for the mdi.

[MDILOCATION target mdiname
    DATANAME mdi partition 1 name
    ...
    DATANAME mdi partition n name]
    
```

File Attributes

File attributes perform tasks such as assigning a name to a physical file, identifying a class location, or identifying an MDI location. For such tasks, use the following attributes in the file declaration:

Attribute	Synonym	Description
MASTERNAME	MASTER	Is the logical name of the Master File.
DATANAME	DATA	Identifies the physical file name of the data source.
WHERE		Is a WHERE criteria.
LOCATION		Is a class location.
INDEXLOCATION	INDEXLOC	Is an index location.
MDILOCATION	MDILOC	Is an MDI location.

Syntax

How to Create a Fusion Access File

```
{MASTERNAME|MASTER} filename {DATANAME|DATA} dataname WHERE expression ;
LOCATION location {DATANAME|DATA} dataname
{MDILOCATION|MDILOC} mdilocation {DATANAME|DATA} dataname3
[WHERE field operator value [OR value] [AND value]...]
WHERE field FROM value TO value]
```

where:

filename

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

dataname

Is the fully qualified physical file name of the data source, in the syntax native to your operating environment. If no extension, file type, or data set is provided, the default values are .fus, FUSION, or FUSION, respectively.

expression

Is a WHERE criteria.

location

Identifies the location of the class. This value is a one- to eight-character name consisting of letters, digits, and underscores.

mdilocation

Identifies the location of the MDI. This value is a one- to eight-character name consisting of letters, digits, and underscores.

field

Is a field in the data source.

operator

Is one of the following:

EQ	Is equal to.
NE	Is not equal to.
GT	Is greater than.
GE	Is greater than or equal to.
LT	Is less than.
LE	Is less than or equal to.

value

Is a valid value.

Example

Adding Attributes to a Declaration

The following example illustrates additional DATANAME attributes added to the Fusion Access File:

```
MASTERNAME CAR
-*the following part is 1998 data.
  DATANAME C:\ibi\srv43\ffs\catalog\car98.fus
  INDEXLOC CTRYIDX
  DATANAME C:\ibi\srv43\ffs\catalog\ctry98.idx
  LOCATION COMPLOC
  DATANAME C:\ibi\srv43\ffs\catalog\comp98.cls

-*the following part is 1999 data.
  DATANAME C:\ibi\srv43\ffs\catalog\car99.fus
  INDEXLOC CTRYIDX
  DATANAME C:\ibi\srv43\ffs\catalog\ctry99.idx
  LOCATION COMPLOC
  DATANAME C:\ibi\srv43\ffs\catalog\comp99.cls

-*this part is for a partitioned MDI
MDILOCATION CARMD
  DATANAME C:\ibi\srv43\ffs\catalog\carmdl.mdi
  DATANAME C:\ibi\srv43\ffs\catalog\carmd2.mdi
  DATANAME C:\ibi\srv43\ffs\catalog\carmd3.mdi
.
.
.
```

The REBUILD MDINDEX utility needs the partition information in the MDILOCATION attribute in order to update an MDI that is larger than two gigabytes. See Chapter 4, *Database Administration Facilities*, for information about REBUILD MDINDEX.

Describing a Data Source: MASTERNAME and DATANAME

If the corresponding data source has the same file name as the Fusion Master File with the default file type, extension or data set, no Fusion Access File is required for that data source. However, if the data source has a different name from the Fusion Master File, a Fusion Access File is needed to identify the data source file. Use the MASTERNAME and DATANAME attributes to describe the data source.

Syntax

How to Describe a Physical Data Source File in the Access File

```
MASTERNAME filename  
DATANAME dataname
```

where:

filename

Is the logical name of the Fusion Master File.

dataname

Is the fully qualified physical file name of the data source, in the syntax native to your operating environment. If no extension, file type, or data set is provided, the extension default values are .fus, FUSION or FUSION, respectively, depending on the operating platform.

Describing a Separately Stored Section or Class: LOCATION

Any section or class in the data source can be stored in its own separate data source. The Fusion Master File declaration for any separately stored section or class must include a LOCATION attribute that assigns a logical name to the separate file. The corresponding Fusion Access File associates this logical name with the name of the physical data source.

Syntax

How to Describe a LOCATION File in the Access File

```
MASTERNAME filename  
  DATANAME dataname_1  
    LOCATION location  DATANAME dataname_2
```

where:

filename

Is the logical name of the Fusion Master File.

dataname_1

Is a fully qualified physical file name that contains the main data source file, in the syntax native to your operating environment. If no extension, file type, or data set is provided, the default values are .fus, FUSION, or FUSION, respectively.

locationname

Is the logical name of the separately stored section or class, derived from the LOCATION attribute in the relevant Fusion Master File declaration.

dataname_2

Is a fully qualified physical file name that contains the class or section, in the syntax of your operating environment. If no extension, file type, or data set is provided, the default values are .fus, FUSION, or FUSION, respectively.

Describing a Horizontal Partition: WHERE

A horizontal partition is a separate data source containing specific field values in a data source, such as year, date, or region. The Fusion Access File describes how to associate the Fusion Master File with its separate data source files. It can also supply a WHERE criteria for each partition to identify the data values stored in that partition; Fusion uses this criteria to optimize retrieval.

Syntax

How to Describe a Horizontal Partition

```
MASTERNAME filename
  DATANAME dataname_1 [WHERE test_1];
  .
  .
  .
  DATANAME dataname_n [WHERE test_n];
```

where:

filename

Is the logical name of the Fusion Master File.

dataname_1...dataname_n

Are the fully qualified physical file names of the partition files associated with the Fusion Master File, in the syntax of your operating environment.

test_1...test_n

Are logical conditions, terminated with semicolons, that describe the data values in the corresponding partitions. Supported operators are EQ, GT, GE, LT, LE, FROM...TO, AND, and OR. See Chapter 5, *Creating Derived Fields*, for information about expressions.

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

Example

Using Horizontal Partitions

The SALES data source consists of partitions for the years 1997, 1998, and 1999:

```
MASTERNAME SALES
  DATANAME C:\ibi\srv43\ffs\catalog\sales99.fus
  DATANAME C:\ibi\srv43\ffs\catalog\sales98.fus
  DATANAME C:\ibi\srv43\ffs\catalog\sales97.fus
```

The user references the three data sources using the logical name SALES:

```
TABLE FILE SALES
```

Examples

Adding a WHERE Criteria to Partitions

This example provides a predicate that describes the values in each partition:

```

MASTERNAME SALES
  DATANAME C:\ibi\srv43\ffs\catalog\sales99.fus
           WHERE SDATE FROM '19990101' TO '19991231';
  DATANAME C:\ibi\srv43\ffs\catalog\sales98.fus
           WHERE SDATE FROM '19980101' TO '19981231';
  DATANAME C:\ibi\srv43\ffs\catalog\sales97.fus
           WHERE SDATE FROM '19970101' TO '19971231';
    
```

With these predicates in the Fusion Access File, query performance increases dramatically because Fusion examines the record selection criteria in each retrieval request and accesses only the specific partitions needed to satisfy the request. For example, in the following request, Fusion does not access the physical file for sales for 1998:

```

TABLE FILE SALES
PRINT *
WHERE SDATE GE '19980401';
END
    
```

Describing Partitions in an Access File

If the data source contains one partition for the year 1998 and another partition for the year 1999, the Fusion Access File must contain a DATANAME declaration, INDEXLOC CTRYIDX declaration, and LOCATION COMPCLOC declaration for each partition. It also needs one MDILOCATION CARMD declaration for the external index, and this declaration must follow all other declarations for the two partitions. This is illustrated in the following declaration for the CAR Master File:

```

MASTERNAME CAR
-*the following part is 1998 data.
  DATANAME C:\ibi\srv43\ffs\catalog\car98.fus
  INDEXLOC CTRYIDX
    DATANAME C:\ibi\srv43\ffs\catalog\ctry98.idx
    LOCATION COMPCLOC
    DATANAME C:\ibi\srv43\ffs\catalog\comp98.cls
-*the following part is 1999 data.
  DATANAME C:\ibi\srv43\ffs\catalog\car99.fus
  INDEXLOC CTRYIDX
    DATANAME C:\ibi\srv43\ffs\catalog\ctry99.idx
    LOCATION COMPCLOC
    DATANAME C:\ibi\srv43\ffs\catalog\comp99.cls
-*this part is for the MD Index
  MDILOCATION CARMD
    DATANAME C:\ibi\srv43\ffs\catalog\carmd.mdi
    
```

Assigning a Logical Name to an Index: INDEXLOCATION and MDILOCATION

In a Fusion data source, a field can participate in several types of indexes. The Fusion Access File describes how to locate these index files by associating a logical index name that was assigned by the LOCATION attribute in the Fusion Master File to a physical file name.

When an index contains the syntax INDEX ON LOCATION in the Fusion Master File, the INDEXLOCATION attribute in the Fusion Access File associates the logical index name to the index physical file. When the syntax INDEX MD_EXTERNAL LOCATION is used, the MDILOCATION attribute in the Fusion Access File performs this function. The MDILOCATION declaration must follow the declarations for all of the partitions involved in the Fusion Access File for a Fusion Master File.

If the MDI could be larger than two gigabytes, or it is anticipated that new data partitions will be added to the data source as time goes on, or REBUILD/MDINDEX/ADD are needed, the MDI must be partitioned into multiple index files. In this case, additional DATANAME attributes must be added to the MDILOCATION CARMD declaration.

Example

Describing an Index Location

Consider the following Fusion Master File:

```
FILENAME CAR FILETYPE FUSION
  CLASS ORIGIN
    FIELD COUNTRY      INDEX ON          LOCATION CTRYIDX
                      INDEX MD_EXTERNAL  LOCATION CARMD
  CLASS COMPANY
    FIELD CAR          INDEX MD_EXTERNAL  LOCATION CARMD
    .
    .
  CLASS BODY
    FIELD BODYTYPE    INDEX MD_EXTERNAL  LOCATION CARMD
```

This Fusion Master File describes two indexes and one location file. The indexes are CTRYIDX, on the COUNTRY field, and CARMD on the COUNTRY, CAR, and BODYTYPE fields. Although the CTRYIDX index is not external and is maintained as part of the data source (INDEX ON), it is stored as a separate file (LOCATION CTRYIDX).

If the data source is not partitioned, the Fusion Access File needs an INDEXLOC declaration for the CTRYIDX index, a LOCATION declaration for the COMPANY class, and an MDILOCATION declaration for the CARMD multi-dimensional index:

```
MASTERNAME CAR
  DATANAME fully qualified physical file name for car
  INDEXLOC CTRYIDX
  DATANAME fully qualified physical file name for ctryidx
  LOCATION COMPLOC
  DATANAME fully qualified physical file name for comploc
MDILOCATION CARMD
  DATANAME fully qualified physical file name for carmd
```

This Fusion Access File will not allow additions to the MDI because this MDI, CARMD, is not partitioned.

Describing Joined Files

The Fusion Master File can describe joins between classes. In simple cases, the Fusion Master File alone may be sufficient for describing the join. However, when the join files are horizontally partitioned, the Fusion Access File is needed in order to implement the join.

If the source class in the join is partitioned but the target class is not, use the instructions in *Describing a Horizontal Partition: WHERE* on page 2-37 to describe the source class. You can use the instructions in *Adding Attributes to a Declaration* on page 2-34 to describe the target class.

For example, if the partitioned SALES file is joined to the non-partitioned CUSTOMER file, the Fusion Access File declarations to describe the joined files are:

```
MASTERNAME SALES
  DATANAME fully qualified physical file name for sales1999
  DATANAME fully qualified physical file name for sales1998
  DATANAME fully qualified physical file name for sales1997
MASTERNAME CUSTOMER
  DATANAME <fully qualified physical file name for customer>
```

Recall that the target field in a join must be indexed. If the source class is partitioned, the target class must either contain the same number of partitions as the source class or only one partition.

```

MASTERNAME SALES
  DATANAME fully qualified physical file name for north east sales
  DATANAME fully qualified physical file name for midwest sales
  DATANAME fully qualified physical file name for southern sales
  DATANAME fully qualified physical file name for west coast sales
  .
  .
  .

MASTERNAME CUSTOMER
  DATANAME fully qualified physical file name for north east customers
  DATANAME fully qualified physical file name for midwest customers
  DATANAME fully qualified physical file name for southern customers
  DATANAME fully qualified physical file name for west coast customers
  
```

However, if the nature of the data source does not have the same number of source partitions as target partitions, then MDI is the only solution to join them together. In the following example, CUSTIX is the MDI used to join SALES to CUSTOMER.

```

MASTERNAME SALES
  DATANAME fully qualified physical file name for sales1999
  DATANAME fully qualified physical file name for sales1998
  DATANAME fully qualified physical file name for sales1997
  .
  .
  .

MASTERNAME CUSTOMER
  DATANAME fully qualified physical file name for region 1
  DATANAME fully qualified physical file name for region 2
  .
  .
  .

-*****
-* MD index file next:
-*****
  MDILOCATION CUSTIX
  DATANAME fully qualified physical file name cusidx
  
```

CHAPTER 3

Using Fusion's High-Performance Features

Topics:

- Creating a Multi-Dimensional Index
- Querying a Multi-Dimensional Index
- Using AUTOINDEX to Choose a Multi-Dimensional Index
- Joining to a Multi-Dimensional Index
- Encoding Values in a Multi-Dimensional Index
- Creating Horizontal and Vertical Partitions
- Using a Vertical Partition

Fusion provides high-performance features that give your data source more functionality. You can utilize a multi-dimensional index and create partitions in your data source.

Note: All sample Access Files in this chapter use the paths that are valid for Windows NT.

Creating a Multi-Dimensional Index

A multi-dimensional index gives complex queries high-speed access to combinations of dimensions across data sources. If you know what information users want to retrieve and why, you can make intelligent choices about index dimensions.

A Fusion Master File can define more than one MDI. If the Fusion Master File defines multiple MDIs, the AUTOINDEX facility chooses the best index to use for each query. For information, see *Using AUTOINDEX to Choose a Multi-Dimensional Index* on page 3-6.

Choosing an Indexing Strategy

The first step in designing an MDI is to find out what kind of information users need from the data source. You should also discuss the availability of system resources with your systems personnel before deciding on an indexing strategy.

You can get advice about your MDIs directly from Fusion. For information, see *Guidelines for a Multi-Dimensional Index* on page 3-3.

Choosing Dimensions for Your Index

The choice of index dimensions depends on knowing the data and on analyzing what is needed from it. Examine the record selection (IF and WHERE) tests in your queries to see how many index dimensions each application actually uses to select records. If different applications need different subsets of dimensions, consider separate MDI's for the separate subsets.

Although Fusion has demonstrated high-speed reporting performance with indexes of up to 30 dimensions, smaller indexes usually generate less retrieval overhead. You can create an unlimited number of MDIs.

The following are good candidates for dimensions in an MDI:

- Fields used frequently in record selection tests. Multiple fields used in selection tests within one query belong in the same MDI.
- Fields used as the basis for horizontal partitioning, such as date or region.
- Derived dimensions that define a new category based on an existing category; for example, if your data source contains the field STATE but you need region for your analysis, you can use the STATE field to derive the REGION dimension.
- Fields with many unique values. Fields which have few possible values are not normally good candidates. However, you may want to index such a field if the data source contains very few instances of one of the values, and you want to find those few instances.

- Packed decimal fields may be used as MDI dimensions on all platforms.
- An MDI can include a field that has a B-tree index as a dimension.

Fusion can also advise you on selecting MDI dimensions.

Reference

Guidelines for a Multi-Dimensional Index

The following guidelines apply to each MDI:

- The maximum size of an MDI is 200 GB.
- The maximum size of each index partition is 2 GB.
- The total size of all dimensions in an MDI cannot exceed 256 bytes. However, if you include the MAXVALUES attribute in the Fusion Master File declaration for a dimension, Fusion uses a small number of bytes to store the values of that dimension:

MAXVALUES	Number of Bytes Required
1 through 253	1
254 through 65,533	2
Greater than 65,533	4

To allow for expansion, if the maximum number of values is close to a limit, make MAXVALUES big enough to use a larger number of bytes. For example, if you have 250 values, specify 254 for MAXVALUES, and reserve 2 bytes for each dimension value.

For information about index attributes in the Fusion Master File, see Chapter 2, *Understanding Your Fusion Schema*.

Using a Multi-Dimensional Index With a Query

Fusion allows you to use an MDI in a TABLE or SQL query.

- There are two ways to use an MDI with a TABLE query:
 - Lazy OR parameters. For example:


```
IF (or WHERE) field EQ value_1 OR value_2 OR value_3. . .
```
 - Mask or wildcard characters. For example, the following would show all values that begin with A for the field FIELD:

```
IF (or WHERE) field EQ A*
```

field is a dimension in an MDI.

- You can use an MDI with an SQL query by issuing an SQL SELECT statement with a WHERE test using a field of an MDI. For example,

```
SELECT field_1, field_2 FROM table WHERE field_3 = value;
```

where *field_3* is a dimension in an MDI.

Note: AUTOINDEX must be turned on for this feature to be operational (see *Using AUTOINDEX to Choose a Multi-Dimensional Index* on page 3-6).

Querying a Multi-Dimensional Index

Fusion provides a query command that generates statistics and descriptions for your MDIs.

The Fusion command ? MDI allows you to display information about MDIs for a given Fusion Master File that hosts the target of your MDI.

Syntax

How to Query a Multi-Dimensional Index

```
? MDI mastername {mdiname|*} [PCHOLD [AS holdfile]]
```

where:

mastername

Is the logical name of the Fusion Master File. If you do not include any other parameters, a list of all MDI names specified is displayed with the command TARGET-OF inside *mastername*. If *mastername* does not have any MDI information, an error message will display.

mdiname

Is the logical name of an MDI. Specifying this parameter displays all the dimensions that are part of this MDI.

mdiname must be specified as TARGET_OF inside *mastername*, or an error message will display. If any of the dimensions are involved in a parent-child structure, a tree-like picture will display.

*

Displays a list of all dimensions, by MDI, whose targets are specified inside *mastername*.

PCHOLD

Saves the output in a default text file, PCHOLD.FTM, and sends it to the client.

holdfile

Is the file in which the output is saved. If this is not included with the AS phrase, the file is named PCHOLD.FTM.

Example

Querying a Multi-Dimensional Index

The following examples illustrate various queries and their results:

- Querying the Fusion Master File ACC43

```
? MDI ACC43
```

returns:

```
XMDI43X1
XMDI43X2
```

XMDI43X1 and XMDI43X2 are targets specified in the Fusion Master File ACC43.

- Querying the XMDI43X2 MDI with Fusion Master File ACC43

```
? MDI ACC43 XMDI43X2
```

returns:

```
S_ORDERS.O_CUSTKEY {300}
L_LINE.L_PARTKEY
```

O_CUSTKEY and L_PARTKEY are dimensions that are part of the MDI XMDI43X2. S_ORDERS and L_LINE are the classes to which the dimensions belong. 300 (in brackets) is the MAXVALUES value for that dimension.

- Querying a Fusion Master File with the * character

```
? MDI ACC43 *
```

returns:

```
XMDI43X1
  1. L_LINE.L_QUANTITY
     2. S_ORDERS.O_CUSTKEY {300}
L_LINE.L_SHIPDATE
XMDI43X2
  S_ORDERS.O_CUSTKEY {300}
  L_LINE.L_PARTKEY
```

All dimensions, listed by MDI, have targets that are specified inside ACC43. L_QUANTITY and O_CUSTKEY are part of a parent-child relationship.

Using AUTOINDEX to Choose a Multi-Dimensional Index

When a Fusion Master File defines multiple MDIs, retrieval efficiency for a query may become a function of the index it uses to access the data source. The Fusion AUTOINDEX facility carefully analyzes each retrieval request and chooses the MDI that provides the best access to the data.

If the selection criteria in a request does not involve any MDI fields, Fusion looks for an appropriate B-tree index to use for retrieval.

You can set the AUTOINDEX command in a stored procedure or in your EDA profile. The AUTOINDEX facility can be enabled or disabled. The default depends on the byte architecture of the operating system being used. If the operating system being used possesses normal byte architecture (for example, HP-UX, MVS, AIX) the default is to enable AUTOINDEX. If the operating system possesses a reverse byte architecture (for example, Digital UNIX, Windows NT), the default is to disable AUTOINDEX.

Reference

Analyzing a Retrieval Request With AUTOINDEX

In its analysis, AUTOINDEX considers the following factors:

- The class most involved in the query.
- The MDI with the most filtering conditions (IF/WHERE selection tests).
- The percent of index dimensions involved in the request from each MDI.
- How close the fields being retrieved are to the target class.
- The size of each MDI.

If everything else is equal, Fusion uses the first MDI it finds in the Fusion Master File.

Syntax

How to Set AUTOINDEX

```
SET AUTOINDEX {ON|OFF}
```

where:

ON

Optimizes MDI retrieval. AUTOINDEX can only be set to ON when the ALL parameter is set to OFF. This value is the default when the operating system possesses normal byte architecture.

OFF

Disables the AUTOINDEX facility. No MDI will be used for retrieval. This value is the default when the operating system possesses reverse byte architecture.

Note: FOCUS TABLE requests can automatically turn off AUTOINDEX and select the appropriate index for retrieval by indicating the index name in the request itself. For example:

```
TABLE FILE filename.mdiname
```

You can also assure access with a specific MDI by creating a Fusion Master File that describes only that index.

Joining to a Multi-Dimensional Index

Joining to an MDI utilizes the power of the MDI to produce a fast, efficient join. Instead of joining one field in the source file to an indexed field in the target file, you can join to multiple dimensions of the MDI.

When the join is implemented, the answer set from the source file is created, and the values retrieved for the source fields serve as index values for the corresponding dimensions in the target MDI.

You can join to an MDI in two ways:

- Join to all dimensions of a named MDI (MDI join). In the MDI join, you pick the MDI to use in the join.
- Join certain dimensions in a dimensional join. In this type of join, the JOIN engine picks the most efficient MDI based on the dimensions you choose. The dimensional join supports partial joins.

Choosing Source Fields

The source fields must form a one-to-one correspondence with the target dimensions. The MDI engine uses the source field values to get pointers to the target class of the MDI, expediting data retrieval from the target file.

You can think of the source fields as mirror dimensions. If you put tighter constraints on the mirror dimensions, a smaller answer set is retrieved from the source file, and fewer Index I/Os are required to locate the records from the target file. The speed of the join improves dramatically with the speed of retrieval of the source file answer set. Therefore, you can expedite any TABLE request against the source file by incorporating selection criteria on fields that participate in either a B-tree or MDI.

The following formula computes the time for a TABLE request that uses an MDI Join:

```
Total Time = Time to Retrieve the answer set from the source file (Ts)
              + Time to retrieve the MD index pointers (Tp)
              + Time to retrieve data from the target file (Tt)
```

Using a B-tree index or MDI in data retrieval reduces all types of retrieval time, reducing the total retrieval time.

Syntax

How to Join to All Dimensions of a Multi-Dimensional Index

```
JOIN field_1 [AND field_2 ...] IN sfile [TAG tag_1]
  TO ALL mdiname IN tfile [TAG tag_2][AS joinname]
[END]
```

where:

field_1, field_2

Are the join fields from the source file.

sfile

Is the logical name of the source file.

tag_1, tag_2

Are one- to eight-character names that can serve as unique qualifiers for field names in a request.

mdiname

Is the logical name of the MDI, built on *tfile*, to use in the join.

tfile

Is the logical name of the target file.

joinname

Is a one- to eight-character join name. A unique join name prevents a subsequent join from overwriting the existing join and allows you to selectively clear the join, and serves as a prefix for duplicate field names in a recursive join.

END

Is required to terminate the JOIN command if it is longer than one line.

Syntax

How to Create a Dimensional Join

```
JOIN field_1 [AND field_2 ...] IN sfile [TAG tag_1]  
    TO ALL dimension_1 [AND dimension_2 ...] IN tfile [TAG tag_2] [AS  
joinname]  
[END]
```

where:

field_1, field_2

Are the join fields from the source file.

tag_1, tag_2

Are one- to eight-character names that can serve as unique qualifiers for field names in request.

sfile

Is the logical name of the source file.

dimension_1, dimension_2

Are dimensions in the target file.

tfile

Is the logical name of the target file.

joinname

Is a one- to eight-character join name. A unique join name prevents a subsequent join from overwriting the existing join and allows you to selectively clear the join, and serves as a prefix for duplicate field names in a recursive join.

END

Is required to terminate the JOIN command if it is longer than one line.

Reference

Guidelines for Choosing a Source Field (Dimensional Joins Only)

- A maximum of four mirror and MDI dimensions can participate in a JOIN command.
- The target of the MDI must be a real class in the target file.
- The order of the mirror dimensions must match the exact physical order of the MDI (target) dimensions.
- The format of each mirror dimension must be identical to that of the corresponding MDI dimension.
- The ALL attribute is required.

Example

Joining to a Multi-Dimensional Index

This example implements an MDI Join from EMPLOYEE to SALE:

```
JOIN EMPID AND DATE AND PRODID IN EMPLOYEE TO ALL SIMDI IN SALE AS A
```

This join is legal because the following join rules have been followed:

- EMPID, DATE, and PRODID map one-to-one onto the ID, DATE, and PRODUCT dimensions in index SIMDI.
- The formats of the fields are identical.

The next join retrieves at least as many records as the previous join because this join uses only two dimensions. If non-unique employee IDs exist in the data source, spurious joins may result:

```
JOIN DATE AND EMPID IN EMPLOYEE TO ALL DATE AND ID IN SALE AS B
```

Encoding Values in a Multi-Dimensional Index

Fusion encodes indexed values any time a field or dimension of an MDI has a MAXVALUES attribute specified or is involved in a parent-child relationship.

Encoded values are stored in the MDI file at rebuild time and can be retrieved and decoded with a TABLE request that specifies the MDIENCODING command. The MDIENCODING command allows the user to get output from the MDI file itself without having to read the FUSION data source. See Chapter 4, *Database Administration Facilities*, for information on the MDIENCODING command.

Reference

Rules for Encoding a Multi-Dimensional Index

The following two rules apply to verb objects in a TABLE request that uses MDIENCODING:

- Only one MDI can be referred to at a time.
- Only dimensions that are part of the same parent-child hierarchy can be used simultaneously in a request. A dimension that is not part of a parent-child relationship can be used as the only verb object in a request if it has a MAXVALUES attribute.

Syntax

How to Encode a Multi-Dimensional Index

```
TABLE FILE mastername.mdiname
request
ON TABLE SET MDIENCODING ON
END
```

where:

mastername

Is the logical name of the Fusion Master File.

mdiname

Is the logical name of the MDI.

request

Is the TABLE request that decodes the MDI.

Example

Encoding a Multi-Dimensional Index

The following provides examples of correct and incorrect MDI encoding syntax.

Correct MDI encoding:

```
TABLE FILE COMPANY.I_DATA1
PRINT CITY BY STATE
ON TABLE SET MDIENCODING ON
END
```

```
TABLE FILE COMPANY.I_DATA1
COUNT CITY
IF STATE EQ NY
ON TABLE SET MDIENCODING ON
END
```

```
TABLE FILE COMPANY.I_DATA1
PRINT CATEGORY
ON TABLE SET MDIENCODING ON
END
```

Incorrect MDI encoding:

```
TABLE FILE COMPANY.I_DATA1
PRINT CITY BY STATE
IF STATE EQ NY
IF CATEGORY EQ RESTAURANT
ON TABLE SET MDIENCODING ON
END
```

This example is incorrect because CATEGORY is not part of the CITY-STATE hierarchy.

Creating Horizontal and Vertical Partitions

Horizontal and vertical partitions offer extensive opportunities for minimizing system resource utilization and overhead.

Using a Horizontal Partition

A horizontal partition can be used for any dimension for which a separate copy of the data source makes sense. For example, if each regional office of a company has its own local data source, you can create a national data source composed of the regional partitions.

A horizontal partition provides a transparent way to handle historical data in a data warehouse or OLAP environment. When used to store historical data, each partition is a snapshot of the entire data source for a specific time period, and updating the data source becomes the process of cleansing and adding another snapshot to the existing data.

You can use a combination of dimensions to define a partition; for example, year and region can combine to create a national data source that contains historical data for each region.

Describing a Horizontal Partition

In a horizontally partitioned data source, each separately stored section, class, or internally maintained index is also partitioned. The Fusion Master File and Access File must list all the location files.

Syntax

How to Describe a Location in a Partitioned Data Source

In the Fusion Master File

```
FILENAME filename ACCESSFILE accessname
[ {CLASS|SECTION} ] name_1 ... [LOCATION loc_1]
[ {CLASS|SECTION} ] name_n ... [LOCATION loc_n]
```

where:

filename

Is the logical name of the data source.

accessname

Is the logical name of the Fusion Access File.

name_1...name_n

Are the names of the classes or sections.

loc_1...loc_n

Are the logical names of the files that contain the class or section data.

In the Fusion Access File

```
MASTERNAME filename
  DATANAME maindat1
    LOCATION loc_1 DATANAME dloc_11
    .
    .
    .
    LOCATION loc_n DATANAME dloc_1n
  DATANAME maindat_2
    LOCATION loc_1 DATANAME dloc_21
    .
    .
    .
    LOCATION loc_n DATANAME dloc_2n
    .
    .
    .
```

where:

filename

Is the logical name of the Fusion Master File.

maindat_1

Is the fully qualified file name of the physical main data source for the first partition.

loc_1...loc_n

Are the LOCATION names of the separately stored classes and sections from the relevant Fusion Master File declarations.

dloc_11...dloc_1n

Are the fully qualified file names of the physical data sources for all separately stored classes and sections from the first partition, in the syntax of your operating environment.

maindat_2

Is the fully qualified file name of the physical main data source for the second partition.

dloc_21...dloc_2n

Are the fully qualified file names of the physical data sources for all separately stored classes and sections from the second partition, in the syntax of your operating environment.

Example**Describing a Location in a Partitioned Data Source**

In the SALES Fusion Master File, the CUSTDATA class is stored in a separate LOCATION file named MORECUST:

```
FILENAME SALES ACCESSFILE XYZ
CLASS SALEDATA
.
.
.
CLASS CUSTDATA LOCATION MORECUST
```

The corresponding Fusion Access File (XYZ.acx) describes one partition for 1999 data, and another partition for the 1998 data. Each partition has its corresponding MORECUST LOCATION file:

```
MASTERNAME SALES
    DATANAME C:\ibi\srv43\ffs\catalog\sales99.fus
    LOCATION MORECUST
    DATANAME C:\ibi\srv43\ffs\catalog\more99.cls
    DATANAME C:\ibi\srv43\ffs\catalog\sales98.fus
    LOCATION MORECUST
    DATANAME C:\ibi\srv43\ffs\catalog\more98.cls
```

The location files within each data source must follow the corresponding main data source file. The main data source file has no LOCATION attribute.

Further, class or section locations may map one-to-one to horizontal partitions, or there may be a single class or section location file for all the horizontal partitions. In the latter case, the location is specified after the last horizontal partition.

Using a Vertical Partition

A vertical partition offers an opportunity to balance system resources based on function. To create a vertical partition intelligently, you must gather information about each data field and its associated applications prior to the data source design stage. The primary uses for vertical partitions are:

- Storing related data together so it is retrieved with minimal overhead.
- Storing unrelated data separately so it can be retrieved separately, to minimize buffer and memory overhead.
- Storing infrequently used data in a separate file, so it is not retrieved unless specifically required, minimizing overhead for the majority of queries.

CHAPTER 4

Database Administration Facilities

Topics:

- Populating a Fusion Data Source: MODIFY FIXFORM
- Migrating Data to Fusion: HOLD FORMAT FUSION
- Migrating a Master File to a Fusion Master File: REBUILD MIGRATE
- Migrating a FOCUS Data Source to Fusion: REBUILD MIGRATE
- Writing a New Partition in a Fusion Access File: HOLD FORMAT FUSION
- Building and Maintaining a Multi-Dimensional Index
- Providing Information About a Fusion Data Source
- Using a Non-Intrusive External Index
- Rebuilding Data Sources Larger Than Two Gigabytes
- Using SmartLoad to Update Multiple Partitions

Database administration facilities allow you to work with data sources and multi-dimensional indexes. Many data source administration tasks can be performed directly on the Fusion/DB engine by issuing command syntax in a stored procedure. For complete information about stored procedures, see the *EDA Stored Procedures Reference Manual*.

Populating a Fusion Data Source: MODIFY FIXFORM

You can populate a Fusion data source by running a simple MODIFY procedure that both transforms and loads the data.

Example

Populating a Fusion Data Source

This example loads input data from a fixed format file with the following record layout:

Field	Length
EMP_ID	9
LAST_NAME	20
FIRST_NAME	10
CURR_SAL	8
CURR_JOBCODE	3

Assume that employees with certain job codes received an increase since the input file was created. The following procedure creates the EMPLOYEE data source from the input data. The numbers on the left refer to the notes that explain the code.

```
1. FILEDEF EMPDATA DISK fully qualified physical filename for data
2. CREATE FILE EMPLOYEE
   MODIFY FILE EMPLOYEE
3. CHECK OFF
4.  FIXFORM  EMP_ID/A9  LAST_NAME/A20  FIRST_NAME/A10  CURR_SAL/A8
      CURR_JOBCODE/A3
5.  DATA ON EMPDATA
6.  END
```

1. The FILEDEF command names the file EMPDATA, and allocates it to the specified directory.
2. The CREATE FILE command creates a Fusion data source using the Fusion Master File named EMPLOYEE.
3. The CHECK OFF command optimizes the loading of the data source.
4. The FIXFORM command describes the format of incoming records.
5. The DATA command identifies the file that contains the input data.
6. The END command terminates the MODIFY procedure.

Migrating Data to Fusion: HOLD FORMAT FUSION

To load a Fusion data source with selected data from an existing data source, execute a stored procedure that queries the data source and includes the ON TABLE HOLD command with the FORMAT FUSION option. This procedure requires the use of an ACCESSFILE or DATASET attribute to indicate the location where the data source should reside.

When a Fusion session ends, any Master Files created by HOLD FORMAT FUSION and REBUILD-MIGRATE are deleted. To avoid this, the HOLDMAST environment variable on Windows NT and UNIX should be set, and allocated on MVS.

- On Windows NT and UNIX, the environment variable can be set to any existing path on the server machine.
- On MVS, the HOLDMAST environment variable should be allocated to an existing PDS.

If HOLDMAST is not set or allocated, an error message is returned and the process terminates.

Syntax

How to Migrate Data With the DATASET Attribute

```
TABLE FILE ...
.
.
.
ON TABLE HOLD [AS filename ] FORMAT FUSION
  DATASET dbfilename [INDEX field_1 ... field_n]
```

where:

filename

Is the name of the Fusion Master File that is created as a result of this request. If omitted, the default file name is HOLD.

dbfilename

Is the fully qualified physical file name for the data source file. The name cannot exceed 72 characters and must be enclosed in single quotation marks if it contains embedded blanks.

INDEX field_1 ... field_n

Creates standard indexes on the fields listed. These fields must be referenced in the query; they will have an INDEX ON attribute in the Fusion Master File.

Syntax

How to Migrate Data With the ACCESSFILE Attribute

```
TABLE FILE ...  
.  
.  
.  
ON TABLE HOLD [AS filename ] FORMAT FUSION  
ACCESSFILE accfilename [INDEX field_1 ... field_n]  
[MD_EXTERNAL dim_1 dim_2 [... dim_n] LOCATION locationname ]
```

where:

dbfilename

Is the fully qualified physical file name for the data source file. The name cannot exceed 72 characters and must be enclosed in single quotation marks if it contains embedded blanks.

accfilename

Is the name of the Fusion Access File to use with this data source. This Fusion Access File must already exist. If the file specifies multiple partitions, the first one will be used for the data. The Fusion Access File name cannot exceed eight characters.

MD_EXTERNAL dim_1 dim_2 [... dim_n]

Creates or updates an MDI, and names at least two dimensions to participate in it. Only one MDI can be named in the HOLD FORMAT FUSION syntax. The dimensions must be verb objects in the query that create the Fusion file; in the Fusion Master File, they have the attribute:

```
INDEX MD_EXTERNAL LOCATION locationname
```

Note: The order of MDI options must match the order shown in the syntax.

LOCATION locationname

Indicates the logical name of the MDI.

Example

Migrating Data to Fusion

The following example illustrates a two segment query:

```
TABLE FILE CAR
SUM CAR AND WIDTH
BY MODEL
LIST CAR
BY MODEL
BY CAR
ON TABLE HOLD FORMAT FUSION ACCESSFILE CAR
  MD_EXTERNAL MODEL WIDTH
  LOCATION MDILOC
  INDEX MODEL
  AS TWOSEG
END
```

The next example illustrates a three segment query:

```
TABLE FILE CAR
SUM CAR AND WIDTH
BY MODEL
SUM CAR
BY MODEL
BY CAR
LIST LENGTH
BY MODEL
BY CAR
BY COUNTRY
ON TABLE HOLD FORMAT FUSION ACCESSFILE CAR
  INDEX COUNTRY
  MD_EXTERNAL MODEL WIDTH CAR
  LOCATION MDILOC
  AS MY3SEG
END
```

Migrating a Master File to a Fusion Master File: REBUILD MIGRATE

You can migrate a Master File to a Fusion Master File by using the REBUILD MIGRATE command. When you use this command, you must identify the name of a Master File that you want to migrate. The REBUILD MIGRATE utility then prompts you to name the Fusion Master File it will create. File names cannot be longer than eight characters.

Reference

Considerations for Migrating a Master File

You must be aware of the following considerations when you migrate a Master File:

- If the Master File you want to migrate includes any DBA commands, they should be removed before starting the migration process.
- You can include all the commands and options needed for all the steps in a stored procedure. The stored procedure should contain each required command or option on a separate line.

Procedure

How to Migrate a Master File to a Fusion Master File

The following illustrates how to migrate a Master File to a Fusion Master File.

1. Issue REBUILD from the prompt.
2. A message displays asking for the REBUILD option. Enter MIGRATE.
3. You are prompted for the SOURCE Master File name to migrate to FUSION. Enter the Master File you want to migrate.
4. A message displays asking for the TARGET Fusion Master File name. Enter the name of the new Fusion Master File.
5. You are asked to verify the migration. Select Y.

The following message appears:

```
Processing MIGRATE request . . .  
Converting SOURCE Master File filename to Fusion Master File Fusionfilename  
...  
FUSION MASTER FILE HAS BEEN SUCCESFULLY CREATED
```

If the SOURCE Master File is a FOCUS data source, REBUILD asks you if you wish to automatically migrate the data from FOCUS to Fusion. For more information on migrating a Fusion data source to Fusion, see *Migrating a FOCUS Data Source to Fusion* on page 4-7.

Migrating a FOCUS Data Source to Fusion: REBUILD MIGRATE

While the HOLD FORMAT FUSION option is convenient for migrating selected portions of a data source to Fusion, the REBUILD utility is more efficient and specific for migrating an entire FOCUS data source to Fusion.

You can migrate a FOCUS data source to a Fusion data source using the MIGRATE option of the REBUILD command. When you use the REBUILD MIGRATE option, you must identify the name of a FOCUS Master File that you want to migrate. The REBUILD MIGRATE utility then prompts you for a name for the Fusion Master File it will create. File names cannot be longer than eight characters.

Reference

Considerations for Migrating a Data Source

You must be aware of the following considerations when you migrate a data source:

- If the Master File you want to migrate includes any DBA commands, they should be removed before starting the migration process.
- If the Master File you want to migrate includes cross references to other FOCUS data sources, you must migrate the cross-referenced files first in order to avoid generating warning messages or errors.
- You can create a Fusion Access File in advance for the new Fusion data source you are creating, and then supply the name of this Fusion Access File to REBUILD MIGRATE.
- You can include all the commands and options needed for all the steps in a stored procedure. The stored procedure should contain each required command or option on a separate line.

Reference

Environment Variables Required to Run REBUILD MIGRATE

To run REBUILD MIGRATE, the user must have the following environment variables set for EDA or FOCUS.

EDA

Environment Variable	Purpose
EDASYNM	Specifies where the created Master File should be saved. If EDASYNM is not defined, EDATEMP or HOLDMAST issued, depending on the operating platform.
EDATEMP (UNIX/Windows NT)	Is a temporary area to hold Master Files. The contents of this area are purged once the EDA session terminates.
HOLDMAST (Windows NT, UNIX, MVS)	Is the environment variable or data set designated for Master Files that are created with the HOLD command.

FOCUS

Environment Variable	Purpose
MASTER (MVS)	Specifies the location in which to save the Master File. If MASTER is not defined, then HOLDMAST will be used. HOLDMAST is the data set used for Master Files that are created with a HOLD command.
TEMPDISK (VM)	Is the disk defined in the FOCUS environment where Master Files created with a HOLD command are stored.
HOLDMAST (Windows NT, UNIX, MVS)	Environment variable or data set designated for Master Files that are created with the HOLD command.

Procedure

How to Run a REBUILD MIGRATE

Note: It is recommended that you create a Fusion Access File for the new Fusion data source before starting the REBUILD MIGRATE.

To run a REBUILD MIGRATE, complete the following steps:

1. Issue the REBUILD command.
2. Select the MIGRATE option in response to a prompt from the REBUILD utility.
3. Identify the FOCUS Master File name when prompted for the source Master File.
4. Identify the name of the target Fusion Master File.
5. You are asked if you are using an Access File.
 - If you created an Access File, select Y and enter the name of the file.
 - If you answered N for having an Access File, you must enter a fully qualified data source file name to use with the DATASET attribute of the new Master File.
6. You are asked to verify the REBUILD MIGRATE. Enter Y.

Example

Migrating a Data Source With REBUILD MIGRATE

This example illustrates the REBUILD MIGRATE steps needed to migrate the FOCUS JOBFIL data source to the Fusion data source NEWJOBFL. The FOCUS JOBFIL Master File appears as follows:

```
FILENAME=JOBFIL, SUFFIX=FOC
SEGNAME=JOBSEG, SEGTYPE=S1
  FIELD=JOB_CODE, 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, $
```

1. Using a text editor, create a Fusion Access File called JOBACC.
2. Enter the following information for the JOBACC Fusion Access File

```
MASTER NEWJOBFL
  DATANAME filename
```

where:

filename

Is a fully qualified physical file name of the NEWJOBFL Fusion data source.

3. Issue REBUILD from the prompt.

4. You are prompted to enter the REBUILD option. Enter MIGRATE.
5. You are asked to enter the source Master File name to migrate from FOCUS to FUSION. Enter JOBFILE.
6. A message displays asking you to enter the target Fusion Master File name. Enter NEWJOBFL.

The following message appears:

```
Processing MIGRATE request . . .
Converting SOURCE Master File JOBFILE to Fusion Master File NEWJOBFL
...
FUSION MASTER FILE HAS BEEN SUCCESFULLY CREATED
```

7. A message displays asking if you want to automatically populate a Fusion data source for this Fusion Master File. Select Y.
8. A message displays asking whether you want to include a Fusion Access File for the Fusion Master File. Select Y.
9. A message appears asking you to enter a Fusion Access File name. Enter JOBACC.
10. You are asked to verify the migration from JOBFILE to NEWJOBFL. Select Y.

The following message displays:

```
Processing MIGRATE request . . .
Converting FOCUS database file JOBFILE to Fusion database file NEWJOBFL
.
.
.
NEW FILE NEWJOBFL ON date and time
NUMBER OF SEGMENTS INPUT= number
```

A new Fusion Master File named NEWJOBFL is created in the EDASYN directory based on the FOCUS Master File named JOBFILE. This Fusion Master File appears as follows:

```
FILE NEWJOBFL FILETYPE FUSION
CLASS JOBSEG KEYS S1
  FIELD JOBCODE          ALIAS JC          FORMAT A3
      INDEX ON
  FIELD 'JOB_DESC'       ALIAS JD          FORMAT A25
CLASS SECSEG SUBCLASS_OF JOBSEG
  FIELD 'SEC_CLEAR'      ALIAS SC          FORMAT A6
CLASS SKILLSEG PART_OF JOBSEG KEYS S1
  FIELD SKILLS           FORMAT A4
  FIELD 'SKILL_DESC'     ALIAS SD          FORMAT A30
```

If the Fusion Access File does not exist, an error will be issued stating that the file does not exist, and the REBUILD MIGRATE will terminate. If the JOBFILE Master File contains any syntax errors, REBUILD MIGRATE produces an error message. If a Master File named NEWJOBFL already exists, REBUILD MIGRATE issues the following error message:

```
NEWJOBFL already exists, please rename it and rerun REBUILD MIGRATE.
```

Writing a New Partition in a Fusion Access File: HOLD FORMAT FUSION

You can use the HOLD FORMAT FUSION command to write a new partition to an existing Fusion Access File. As part of the query, you can create standard indexes or create a new MDI.

Syntax

How to Write a New Partition in an Existing Access File

```
TABLE FILE ...
.
.
.
ON TABLE HOLD [AS filename ] FORMAT FUSION
    DATANAME dbfilename APPEND_TO ACCESS accfilename
    [INDEX field_1 ... field_n]
    [MD_EXTERNAL dim_1 dim_2 [... dim_n] LOCATION locationname ]
```

where:

dbfilename

Is the fully qualified physical file name for the data source file. The name cannot exceed 72 characters and must be enclosed in single quotation marks if it contains embedded blanks.

accfilename

Is the name of the Fusion Access File to use with this data source. This Fusion Access File must already exist. The file name designated by *dbfilename* will be appended to the bottom of this Fusion Access File and will become the last partition, which will contain the held data. The file name cannot exceed eight characters.

field_1 ... field_n

Are the fields to index.

dim_1 ... dim_n

Are the dimensions that will participate in the MDI.

locationname

Is the location of the MDI.

Reference

Notes on Using HOLD FORMAT FUSION

The following are considerations for using HOLD FORMAT FUSION:

- The number of classes in the Fusion Master File depends on the query; multi-verb requests produce multi-class Fusion Master Files.
- The bottom class of the Fusion Master File will have a `TARGET_OF locationname` attribute, indicating that it is the target of the MDI that will be built.
- A new MDI named *locationname* is created.
- A Fusion file named HOLD is created unless you specify a file name with the AS option. The index file *locationname* is created with an .mdi extension or MDI file type or MDI data set extension.
- You can include INDEX and MD_EXTERNAL options in the same query in any order; however, you can specify each only once.
- INDEX and MD_EXTERNAL are optional. You must include all attributes that are part of an index request:
 - INDEX requires you to specify at least one field to be indexed.
 - MD_EXTERNAL requires you to specify dimensions before the location. You must specify at least two dimensions.
- MD_EXTERNAL and DATASET cannot be used together because MD_EXTERNAL requires its physical file name to be specified in a Fusion Access File. If these attributes are used together in a HOLD FORMAT FUSION request, an error message will result.

If your query violates any syntax rules, an error message is returned and your request is not processed.

Reference

Messages Generated During HOLD FORMAT FUSION Processing

Fusion generates the following messages while processing a valid HOLD FORMAT FUSION request:

PROCESSING QUERY . . .

After the parsing process ends, the data source associated with the master file in your query is read.

HOLDING QUERY OUTPUT . . .

Data is held in a temporary file.

CREATING FUSION FILE *datafile* . . .

The Fusion data source is created. If you did not specify a file name, the file will be named HOLD.

BUILDING MDI FILE *locationname* . . .

The multi-dimensional index is created. Its name is *locationname*, the value of the LOCATION command.

THE QUERY HAS BEEN SUCCESSFULLY PROCESSED . . .

A Fusion data source, a Fusion Master File, and an MDI file have been successfully created. You can run a query against your new Fusion data source.

If your HOLD FORMAT FUSION request violates any rules, you get the appropriate error message followed by the message:

YOUR REQUEST CANNOT BE COMPLETED

Note: To use the command HOLD FORMAT FUSION, the HOLDMAST environment variable must be set on Windows NT or UNIX, and allocated on MVS. See *Migrating a FOCUS Data Source to Fusion* on page 4-7.

Building and Maintaining a Multi-Dimensional Index

The REBUILD command is used to create or maintain a multi-dimensional index. This command can be issued in a Fusion session, or a stored procedure. If issued in a Fusion session, the REBUILD command conducts a dialogue with the user. To issue the REBUILD command in a stored procedure, you place the REBUILD command and the user-supplied portion of the REBUILD dialogue in the stored procedure.

If the MDI file might be larger than two gigabytes or if you plan to add more data partitions to it, the index file must be partitioned from the initial REBUILD phase.

Once the index has been created, you can use it in a retrieval request. You cannot use an MDI for modifying the data source. If you update the Fusion data source without rebuilding the MDI and then attempt to retrieve data with it, Fusion displays a message indicating that the MDI is out of date. You must then rebuild the MDI.

When upgrading to a new release of Fusion, you should rebuild all existing multi-dimensional indexes as NEW. If they are not rebuilt, the server may hang when attempting to use the multi-dimensional index.

Example

Creating a Multi-Dimensional Index in a Fusion Session

This example illustrates the creation of an MDI, XMDI1. The following Fusion Master Files describe the data sources that participate in the XMDI1 MDI:

- ORDERSI1
- CUSTOMER
- PARTS
- SUPPLIER

The top class in the ORDERSI1 data source, class L_LINE, is the TARGET_OF class for the MDI. The ORDERSI1 Master File describes the following joins:

Source Data Source	Source Class	Joined Data Source	Joined Class
ORDERSI1	S_ORDERS	CUSTOMER	C_CUST
ORDERSI1	L_LINE	PARTS	P_PART
ORDERSI1	L_LINE	SUPPLIER	S_SUPP

Appendix A, *Sample Schemas*, shows the Fusion Master Files for these data sources.

The following is the REBUILD procedure for the creation of the MDI.

1. Issue REBUILD at the command prompt.

The REBUILD command invokes the REBUILD utility.

2. The following menu displays:

```
Enter option
 1. REBUILD           (Optimize the database structure)
 2. REORG             (Alter the database structure)
 3. INDEX            (Build/modify the database index)
 4. EXTERNAL INDEX   (Build/modify an external index database)
 5. CHECK            (Check the database structure)
 6. TIMESTAMP        (Change the database timestamp)
 7. DATE NEW         (Convert old date formats to smartdate formats)
 8. MDINDEX          (Build/modify a multidimensional index. FUSION DBs only)
 9. MIGRATE          (Convert FOCUS masters/DBs to FUSION)
```

Enter MDINDEX. The MDINDEX command invokes the MDI option of the REBUILD utility.

3. The following message appears:

```
NEW/ADD
```

Enter NEW. The NEW/ADD option enables you to create a new MDI or to add new data partitions to an existing MDI.

4. The following message appears:

```
ENTER THE NAME OF THE MASTER
```

Enter ORDERSI1. The ORDERSI1 Fusion Master File contains the class that is the TARGET_OF the MDI.

5. The following message appears:

```
ENTER MD_INDEX LOCATION FILE NAME
```

Enter XMDI1. XMDI1 is the logical name of the XMDI1 MDI.

6. The following message appears:

```
ANY RECORD SELECTION TESTS? (YES/NO)
```

Enter NO to indicate that there are no record selection criteria for the index. The user can limit the index to records that meet certain criteria by entering YES, followed by valid record selection expressions, followed by the END command. For information on expressions, see Chapter 5, *Creating Derived Fields*.

The following stored procedure creates the XMDI1 MDI and contains only the user-supplied portion of the preceding dialogue, entered in upper case:

```
REBUILD
MDINDEX
NEW
ORDERSI1
XMDI1
NO
```

For complete information on stored procedures, see the *EDA Stored Procedures Reference Manual*.

Example

Using a Multi-Dimensional Index With a Request

The following TABLE request accesses the ORDERSI1 data source. You know it will use the XMDI1 index for retrieval because XMDI1 is the only MDI described in the Fusion Master File:

```
TABLE FILE ORDERSI1
PRINT L_QUANTITY C_MKTSEGMENT P_TYPE S_REGION PS_SUPPLYCOS
BY O_ORDERDATE
-* WHERE Condition utilizing MDI fields:
WHERE (O_ORDERDATE FROM '19980101' TO '19980131')
      AND (P_TYPE EQ 'ALUMINIUM')
      AND (L_QUANTITY LT 25)
      AND (S_REGION EQ 'EUROPE' OR 'ASIA')
      AND (PS_SUPPLYCOS FROM 200 TO 250) ;
END
```

Partitioning a Multi-Dimensional Index

If the data source has grown due to the addition of new data partitions, and these partitions need to be added to the MDI, you must perform the following steps:

1. Update the Fusion Access File to include the new data partitions.
2. Verify that your MDI is partitioned. Remember that the ADD function of the REBUILD utility cannot be executed on a non-partitioned MDI.
3. Perform the REBUILD, MDINDEX, and ADD on the MDI.

Example

Adding a Partition to a Multi-Dimensional Index

The following procedure is a REBUILD dialogue:

1. Issue REBUILD at the command prompt. The REBUILD command invokes the REBUILD utility.
2. The following menu displays:

```
Enter option
1. REBUILD                (Optimize the database structure)
2. REORG                  (Alter the database structure)
3. INDEX                  (Build/modify the database index)
4. EXTERNAL INDEX        (Build/modify an external index database)
5. CHECK                  (Check the database structure)
6. TIMESTAMP              (Change the database timestamp)
7. DATE NEW               (Convert old date formats to smartdate formats)
8. MDINDEX                (Build/modify a multidimensional index. FUSION DBs
only)
9. MIGRATE                (Convert FOCUS masters/DBs to FUSION)
```

Enter MDINDEX. This command invokes the MDI option of the REBUILD utility.

3. The following message appears:

```
NEW/ADD
```

Enter ADD. The NEW/ADD option enables you to create a new MDI or to add new data partitions to an existing MDI.

4. The following message appears:

```
ENTER THE NAME OF THE MASTER
```

Enter the name of the Master File. The Fusion Master File contains the class that is the TARGET_OF the MDI.

5. The following message appears:

```
ENTER MD_INDEX LOCATION FILE NAME
```

Enter the logical name of the MDI.

6. The following message appears:

```
ANY RECORD SELECTION TESTS? (YES/NO)
```

Enter NO to indicate that there are no record selection criteria for the index. The user can limit the index to records that meet certain criteria by entering YES, followed by valid record selection expressions, followed by the END command. For information on expressions, see Chapter 5, *Creating Derived Fields*.

Once the MDI is rebuilt to include the new data partitions, any retrieval query that uses the MDI will use the newly added data partitions within that MDI.

Querying the Progress of a Multi-Dimensional Index

Use the SET MDIPROGRESS command to view messages about the progress of your MDI build. The messages will show the number of data records accumulated for every n records inserted into the MDI as it is processed.

Syntax

How to Query the Progress of a Multi-Dimensional Index

```
SET MDIPROGRESS = {0| $n$ }
```

where:

n

Is an integer greater than 1000, which will display a progress message for every n records accumulated in the MDI build. If you enter 0, progress messages will not display.

Displaying a Warning Message

The SET MDICARDWARN command displays a warning message every time a dimension's CARDINALITY exceeds a specified value, offering you the chance to study the MDI build. When the number of equal values of a dimension's data reaches a specified percent, a warning message will be issued. In order for MDICARDWARN to be reliable, the data source should contain at least 100,000 records.

Note: In addition to the warning message, a number displays in brackets. This number is the least number of equal values for the dimension mentioned in the warning message text.

Syntax

How to Display a Warning Message

```
SET MDICARDWARN =  $n$ 
```

where:

n

Is a percentage value from 0 to 50.

Retrieving Data Directly From a Multi-Dimensional Index

The SET MDIENCODING command allows you to get data from the MDI file itself without having to read the FUSION data source. This command is available only from within a TABLE request.

The following two rules apply to verb objects in a TABLE request that uses MDIENCODING:

- Only one MDI can be referred to at a time.
- Only dimensions that are part of the same parent-child hierarchy can be used simultaneously in a request. A dimension that is not part of a parent-child relationship can be used as the only verb object in a request only if it has a MAXVALUES attribute.

Syntax

How to Retrieve Data From a Multi-Dimensional Index

```
SET MDIENCODING {ON|OFF}
```

where:

ON

Retrieves data directly from the MDI file.

OFF

Does not retrieve data directly from the MDI file.

Providing Information About a Fusion Data Source

The ? FILE and ? FDT commands display information about the physical structure of a Fusion data source. These commands support Fusion files by displaying information on all the partitions mentioned in the Fusion Access File entry for that Fusion Master File. The ? FILE and ? FDT commands also display information on the status of all Fusion data source files in a Fusion Access File for the specified Fusion Master File.

References

Information Retrieved By ? FDT

The ? FDT command displays the following information:

- Segment (class) names and lengths.
- Parent of the segment (class).
- Starting page, ending page, and total pages.

Information Retrieved By ? FILE

The ? FILE command displays the following attributes:

- Total number of segments (classes).
- Total characters.
- Total pages.

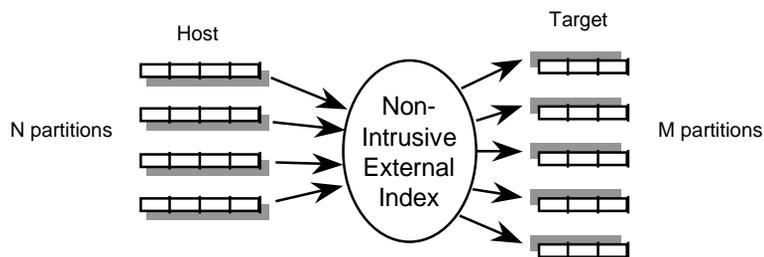
Using a Non-Intrusive External Index

A non-intrusive external index is an index that is created as it is needed. It lives outside the data source and cannot be updated along with the data source. The data source does not know of its existence, and any changes to the target data source are not reflected in the non-intrusive external index until it is rebuilt. A non-intrusive external index can have a maximum of 120 partitions and a maximum size of 2 GB. A non-intrusive external index does not require any modifications to the Fusion Master File or Fusion Access File.

You can create a non-intrusive external index in a Fusion session or stored procedure. You can also create a non-intrusive external index by establishing a JOIN using the indexed field.

A non-intrusive external index should be used in the following situations:

- When you need to index many horizontal partitions (up to 120) by directly mapping to all of the partitions instead of using a standard B-tree index that can only be used with one horizontal partition. A non-intrusive external index is extremely useful when joining data sources that contain several partitions. For example, in the following figure the host data source containing N partitions and the target containing M partitions utilize a non-intrusive external index to join all of the host partitions to all of the target partitions:



- For queries that your application was not intended to perform using non-indexed fields.
- For easy administration. The non-internal external index can be deleted without affecting any application.

- When using a derived field, especially a dynamic derived field, in the Fusion Master File. Internal indexes cannot use derived fields.

Procedure

How to Build a Non-Intrusive External Index in a Fusion Session

Complete the following steps to build a new non-intrusive external index using the interactive dialogue:

1. Issue the REBUILD command and press Enter.
2. The following menu displays:

```
Enter option
 1. REBUILD           (Optimize the database structure)
 2. REORG             (Alter the database structure)
 3. INDEX             (Build/modify the database index)
 4. EXTERNAL INDEX   (Build/modify an external index database)
 5. CHECK             (Check the database structure)
 6. TIMESTAMP        (Change the database timestamp)
 7. DATE NEW         (Convert old date formats to smartdate formats)
 8. MDINDEX          (Build/modify a multidimensional index. FUSION DBs
only)
 9. MIGRATE           (Convert FOCUS masters/DBs to FUSION)
```

Enter EXTERNAL INDEX and press Enter.

3. The following message displays:

```
NEW, OR ADD TO EXISTING INDEX DATABASE? (NEW/ADD)
```

Enter NEW and press Enter.

4. The following message displays:

```
ENTER FILENAME OF EXTERNAL INDEX
```

Enter a logical name for the index and press Enter.

5. The following message displays:

```
ENTER LOGICAL NAME OF FOCUS/FUSION DATABASE TO INDEX
```

Enter the target data source name and press Enter.

6. The following message displays:

```
ENTER NAME OF FIELD TO INDEX
```

Enter the name of the field to index and press Enter.

7. The following message displays:

```
POINT INDEX TO AN ALTERNATE FIELD? (DEFAULT=EXTERNAL INDEX FIELD)
(YES/NO)
```

Enter NO and press Enter.

8. The following message displays:

```
ANY RECORD SELECTION TESTS? (YES/NO)
```

Enter NO and press Enter.

You have now built a non-intrusive external index.

Procedure

How to Build a Non-Intrusive External Index With a Stored Procedure

Complete the following steps to build a non-intrusive external index through the use of a stored procedure:

1. Issue a USE command to establish the link between the data source and the non-intrusive external index with the following syntax

```
USE  
filename  
NEW AS logical_name  
END
```

where:

filename

Is the fully qualified physical file name of the non-intrusive external index.

logical_name

Is the logical name of the non-intrusive external index.

If you do not issue the USE command, the non-intrusive external index will be built in the current directory with the file extension .idx, file type INDEX, or data set extension INDEX.DATA.

2. Create a stored procedure including the REBUILD command and the EXTERNAL INDEX option. The syntax is

```
REBUILD  
EXTERNAL INDEX  
NEW  
indname  
datname  
fieldname  
index  
tests
```

where:

indname

Is the logical name of the external index.

datname

Is the logical name of the FOCUS/Fusion data source to index

fieldname

Is the field to index.

index

Is a Y or N answer to whether to point the index to an alternate field. Answering Y makes the index target the class specified by the alternate field.

tests

Is a Y or N answer to whether record tests should be included.

When the procedure is completed, you will receive statistics about the build.

Procedures

How to Use a Non-Intrusive External Index

Complete the following steps to use the non-intrusive external index.

1. Issue the USE command

```
USE
filename
WITH database
END
```

where:

filename

Is the fully qualified file name of the non-intrusive external index.

database

Is the logical name of the target data source.

2. Execute any query against the target data source that has a WHERE or IF criteria on the indexed field.

How to Add a New Partition to a Non-Intrusive External Index

Complete the following steps to add new partitions to the non-intrusive external index:

1. List the new partition entries in the Fusion Access File for the data source.
2. Issue a USE command to establish the link to the target data source

```
USE
filename
WITH database
END
```

where:

filename

Is the fully qualified file name of the non-intrusive external index.

database

Is the logical name of the target data source.

3. Issue the REBUILD command

```
REBUILD  
EXTERNAL INDEX  
ADD  
indname  
tests
```

```
REBUILD  
EXTERNAL INDEX  
ADD  
filename  
tests
```

where:

filename

Is the logical name of the non-intrusive external index.

tests

Is a Y or N answer to the whether record selection tests should be included.

When the procedure is complete, the non-intrusive external index will be updated to include the new partitions.

Procedure

How to Validate a Non-Intrusive External Index

Complete the following steps to check the validity of a non-intrusive external index:

1. Issue a USE command with the AS option to establish the logical name for the non-intrusive external index

```
USE  
filename  
AS logical_name  
END
```

where:

filename

Is the fully qualified physical file name of the non-intrusive external index.

logical_name

Is the logical name of the non-intrusive external index.

2. Issue the ? FDT command

```
? FDT
```

where:

logical_name

Is the logical name of the non-intrusive external index.

A full description of the non-intrusive external index is displayed.

Rebuilding Data Sources Larger Than Two Gigabytes

Since Fusion data source partitions can contain external class locations, their total size can exceed the REBUILD limit of 2 gigabytes (GB). To use REBUILD options like DUMP, LOAD, and REBUILD on these large files, one must specify multiple directories to store the intermediate files that REBUILD creates. The user may specify up to 64 directories for storing intermediate files. The number of directories required is approximately equal to the total size of the data source divided by 2 GB.

Procedure

How to Specify Directories for Intermediate Files

The following steps assign directories to hold the intermediate files that REBUILD creates:

1. Create the desired number of directories with the following syntax:

UNIX

```
Export IBIREB01=/home1/tempfiles
Export IBIREB02=/home1/user/tempfiles
...
Export IBIREB64=/home3/foo/files
```

Windows NT

```
SET IBIRKB01=\home1\tempfiles
SET IBIRKB02=\home1\user\tempfiles
...
SET IBIRKB64=\home3\foo\files
```

2. Perform the desired REBUILD function. The temporary files will be created as needed and then deleted at the end of the REBUILD.

To deactivate this capability, the user must clear the environment variables or set the environment variables to empty values.

Note: In case of problems with the rebuilding of the data source, it is up to the user to delete the temporary files, rebuild*.ftm, within the additional temporary directories.

Using SmartLoad to Update Multiple Partitions

Note: This feature is available only for FOCUS 7.0.9 and 7.1.

The Fusion SmartLoad feature allows you to update a multiple-partition Fusion data source from one transaction file by generating several TABLE requests that create transaction files. Data from the input file is segregated into separate output files—one for each partition of the data source. You can execute SmartLoad at the FOCUS command line, or from a procedure.

WHERE screening tests, coded for each partition, are evaluated for each output file. When a transaction passes the WHERE screening test for multiple partitions, the transaction is placed in the first appropriate partition. The data is not replicated in multiple partitions. Transactions that do not pass any WHERE screening test are logged to the orphan file under ddname FOC4000. SmartLoad uses these files as transaction input for a user-supplied MODIFY procedure.

Transactions for each partition are created in ddnames FOC4*nnn*, where *nnn* is the 3-digit partition number, zero filled. For example, transactions for the first partition are created under ddname FOC4001. The SmartLoad utility automatically generates Master Files with the same names as the transaction files to describe the transaction files.

Syntax

How to Execute the SmartLoad Procedure

```
EX SMRTLOAD FILE=fusion_mfd,INFILE=transaction_mfd,INFEX=modify_focexec
```

where:

fusion_mfd

Is the Master File for the Fusion data source to be loaded.

transaction_mfd

Is the Master File that describes the input file.

Any simple structure that can be described with a Master File can be used as input for the SmartLoad utility. Simple one-segment structures or single path hierarchies are recommended.

The input file description must support evaluation of the WHERE criteria in the Fusion Access File. Permanent DEFINE commands may be added to the transaction file description to support the WHERE criteria.

modify_focexec

Is the MODIFY FOCEXEC that you want to use to update the Fusion data source *fusion_mfd* with transaction data described by *transaction_mfd*. This MODIFY cannot be compiled.

The MODIFY must describe the input transaction format using the following syntax:

```
FIXFORM FROM transaction_mfd
```

The MODIFY must identify the transaction source using the following syntax:

```
DATA ON transaction_mfd
```

When executing SmartLoad in an online environment, you will be prompted for the required parameters if they are not supplied on the command line:

```
EX SMRTLOAD
ENTER THE NAME OF THE FUSION MASTER fusion_mfd
ENTER THE NAME OF THE INPUT TRANSACTION FILE transaction_mfd
ENTER THE NAME OF THE UPDATE PROGRAM modify_focexec
```

Example

Using SmartLoad

This example illustrates the use of the SmartLoad feature on the ORDERS data source. The following are the files being used with the SmartLoad feature. There is more than one Access File listed because they are specific to the operating system.

- Fusion Master File

```
FILE ORDERS FILETYPE FUSION ACCESSFILE ORDERS
CLASS S-ORDERS KEY S1
FIELD ORDERKEY                FORMAT I9    INDEX ON
FIELD CUSTKEY                  FORMAT I9    INDEX ON
FIELD ORDERSTATU              FORMAT A1
FIELD TOTALPRICE              FORMAT D15.2
FIELD ORDERDATE               FORMAT YYMD
FIELD SHIPPRIORI              FORMAT D10
```

- Fusion Access File ORDERS

MVS

```
MASTER ORDERS
DATANAME USER1.ORDERS1.FUSION
WHERE ORDERDATE FROM '1998/01/01' TO '1998/01/31';
DATANAME USER1.ORDERS2.FUSION
WHERE ORDERDATE FROM '1998/02/01' TO '1998/02/28';
DATANAME USER1.ORDERS3.FUSION
WHERE ORDERDATE FROM '1998/03/01' TO '1998/03/31';
```

VM

MASTER ORDERS

```
DATANAME 'ORDERS1 FUSION A'
  WHERE ORDERDATE FROM '1998/01/01' TO '1998/01/31';
DATANAME 'ORDERS2 FUSION A'
  WHERE ORDERDATE FROM '1998/02/01' TO '1998/02/28';
DATANAME 'ORDERS3 FUSION A'
  WHERE ORDERDATE FROM '1998/03/01' TO '1998/03/31';
```

- Transaction Master File TRANSIN

```
FILE=TRANSIN, SUFFIX=FIX,$
  SEGNAME=ONE,$
    FIELD=ORDERKEY                ,,USAGE=I9, ACTUAL=A9,$
    FIELD=CUSTKEY                  ,,USAGE=I9, ACTUAL=A9,$
    FIELD=ORDERSTATU              ,,USAGE=A1, ACTUAL=A1,$
    FIELD=TOTALPRICE              ,,USAGE=D15.2,
  ACTUAL=Z15.2,$
    FIELD=ORDERDATE              ,,USAGE=YYMD,          ACTUAL=A8,$
    FIELD=SHIPPRIORI            ,,USAGE=D10, ACTUAL=A10,$
```

The following transaction data source, TRANSIN, is evaluated against the WHERE criteria listed in the Fusion Access File:

```
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5--
000000101      123Y          1011119980101      101
000000202      123Y          2022219980201      202
000000303      123Y          3033319980301      303
000000404      123Y          4044419980401      404
```

- The first three transactions each satisfy a WHERE test for one of the partitions, and the fourth transaction does not satisfy any of the WHERE conditions. Therefore, This transaction will be logged in the orphan file (the file allocated to ddname FOC4000).

The following is the MODIFY procedure, UPDTPROG, that will update the partitions:

```
MODIFY FILE ORDERS
  FIXFORM FROM TRANSIN
  MATCH ORDERKEY
    ON MATCH UPDATE CUSTKEY ORDERSTATU TOTALPRICE ORDERDATE SHIPPRIORI
    ON NOMATCH INCLUDE
  DATA ON TRANSIN
  END
```

The SmartLoad utility by run by issuing the following command:

```
EX SMRTLLOAD FILE=ORDERS,INFILE=TRANSIN,INFEX=UPDTPROG
```

This results in the following output. The numbers on the left refer to the explanatory notes that appear after the example:

```

1. NUMBER OF ERRORS=      0
   NUMBER OF SEGMENTS=   1 ( REAL=   1 VIRTUAL=   0 )
   NUMBER OF FIELDS=     6 INDEXES=   2 FILES=   1
   TOTAL LENGTH OF ALL FIELDS=  52
   HOLDING...
2. 1 FILE(S) LOADED
3. NUMBER OF ERRORS=      0
   NUMBER OF SEGMENTS=   1 ( REAL=   1 VIRTUAL=   0 )
   NUMBER OF FIELDS=     6 INDEXES=   0 FILES=   1
   TOTAL LENGTH OF ALL FIELDS=  52
   HOLDING...
4. HOLDING...
   NUMBER OF RECORDS IN TABLE=      1 LINES=      1
5. HOLDING...
   NUMBER OF RECORDS IN TABLE=      1 LINES=      1
6. HOLDING...
   NUMBER OF RECORDS IN TABLE=      1 LINES=      1
7. HOLDING...
   NUMBER OF RECORDS IN TABLE=      1 LINES=      1
8. ORDERS1 FUSION    ON 12/23/1998 AT 15.13.37
   TRANSACTIONS:      TOTAL =      1 ACCEPTED=      1 REJECTED=      0
   SEGMENTS:          INPUT =      0 UPDATED =      1 DELETED =      0
9. ORDERS2 FUSION    ON 12/23/1998 AT 15.13.40
   TRANSACTIONS:      TOTAL =      1 ACCEPTED=      1 REJECTED=      0
   SEGMENTS:          INPUT =      0 UPDATED =      1 DELETED =      0
10. ORDERS3 FUSION   ON 12/23/1998 AT 15.13.41
   TRANSACTIONS:      TOTAL =      1 ACCEPTED=      1 REJECTED=      0
   SEGMENTS:          INPUT =      0 UPDATED =      1 DELETED =      0

```

1. Checks the Fusion data source description for errors.
2. Checks the update MODIFY procedure for existence.
3. Checks the transaction file description for errors.
4. Retrieves the transaction dated 1998/01/01 for the first partition and store it in the file allocated to ddname FOC4001.
5. Retrieves the transaction dated 1998/02/01 for the second partition and store it in the file allocated to ddname FOC4002.
6. Retrieves the transaction dated 1998/03/01 for the third partition and store it in the file allocated to ddname FOC4003.
7. Retrieves the extraneous transaction dated 1998/04/01 that does not fit into any partition and store it in the file allocated to ddname FOC4000. This transaction is not loaded into the Fusion data source.
8. Loads transactions into the first partition.

9. Loads transactions into the second partition.
10. Loads transactions into the third partition.

Reference

SmartLoad Usage Notes

- SmartLoad clears all DEFINES for the file described by *transaction_mfd*.
- Adequate workspace, such as temporary attached disk storage, must be available for temporary files. As a rule of thumb, have space 25 to 50% larger than the size of the original input file available.
- If your MODIFY procedure logs transactions to a file (for example, LOG DUPL ON *ddname*), assign the output file the disposition MOD so that all transactions are appended to the same file. Note that the format of the log file will match the layout created by SmartLoad and described in the Master File named FOC4001.
- The user-supplied MODIFY procedure should contain only MODIFY commands or Dialogue Manager commands that capture or display statistics (-RUN, ? STAT, -TYPE). Under no circumstances should the procedure contain -QUIT, -QUIT FOCUS, or FIN, which would prematurely end the SmartLoad execution.
- Do not use *transaction_mfd* anywhere in the MODIFY procedure, except in the FIXFORM FROM and DATA ON commands. The utility uses LET substitution to identify transaction data for each partition.
- Report and MODIFY statistics are not displayed when SET MESSAGE=OFF is in effect.
- SmartLoad echoes error messages generated by FOCUS or EDA. Captured error message numbers are available in &&SMRTLOAD. Test &&SMRTLOAD for errors when writing procedures using SmartLoad. Note that errors produced during execution of the user-supplied MODIFY procedure are not captured in &&SMRTLOAD.
- To avoid potential problems, clear all LETs and JOINS before executing the SmartLoad utility.

CHAPTER 5

Creating Derived Fields

Topics:

- Using a Derived Field
- Calculating a Derived Field With Missing Values
- Types of Expressions
- Numeric Expressions
- Date Expressions
- Alphanumeric Expressions
- Logical Expressions
- Conditional Expressions

A derived value is calculated by including an expression in the derived dimension's field declaration. An expression is a string of field names, constants, functions, and operators that resolves to a single value. You can use an expression to assign a value to a temporary field or a variable in a stored procedure, or transform data values before storing them in a data source.

The syntax used in this chapter is different from prior releases. However, the old syntax is still supported.

The use of constants and operators is explained in this topic. For more information on functions, see Appendix B, *Fusion Functions and Subroutines*.

Using a Derived Field

A Fusion Master File can describe how to calculate a value that is not stored in the data source. This is a derived value. You can use this derived value as though it were an actual data source field—you can even make it a dimension in a multi-dimensional index (MDI). For more information on MDIs see Chapter 3, *Using Fusion's High-Performance Features*.

Syntax

How to Create a Derived Field

`FIELD name format DEFINED BY expression; [DFC cc YRT nn] [MISSING attributes]`

where:

name

Is the name of the derived field. It can be any name that complies with Fusion field naming conventions.

format

Is the format in which to display the derived field in reports. The default value is D12.2.

expression

Is the expression that defines the value of the derived field. The expression can include any of the following elements: data source fields, constants, arithmetic operators, IF tests, logical operators, and character operators. For complete information about expressions, refer to *Types of Expressions* on page 5-4.

cc

Is a two-digit century.

nn

Is a two-digit year that defines the lowest year for which *cc* applies.

See Chapter 2, *Understanding Your Fusion Schema*, for more information on defining the correct century.

attributes

Are attributes that describe how to calculate the derived field value when fields used in the expression have missing values. For specific attributes see *Calculating a Derived Field With Missing Values* on page 5-3.

Example**Creating a Derived Field**

The following is an example of a derived field. It does not define a default century or a MISSING attribute:

```
FIELD RETAIL_PRICE USAGE D8.2 DEFINED BY (IF COST LT 100 THEN 100 * 1.5
ELSE 100 * 1.25);
```

Calculating a Derived Field With Missing Values

You can use a modified form of the MISSING attribute to manipulate the calculation of a derived field when some of the fields involved contain missing values.

Syntax**How to Calculate a Derived Field With Missing Values**

```
MISSING {OFF|ON} [NEEDS] {SOME|ALL}
```

where:

OFF

Substitutes a zero for a missing numeric field and a blank for a missing alphanumeric field before calculating the field value. This value is the default.

ON

Supports missing values. The SOME and ALL values designate specific behavior.

NEEDS

Signals printed data options.

SOME

Calculates the derived field based on how many missing values exist in the calculation:

- If all field values in the calculation are missing, a missing value is assigned to the derived field.
- If at least one field value in the calculation is present, a zero is substituted for a missing numeric value and a blank for a missing alphanumeric value. The calculation is then completed.

This value is used only when ON is specified. SOME is the default value when MISSING is ON.

ALL

Assigns a missing value to the derived field if *any* value used in the calculation is missing. This value is used only when ON is specified.

Types of Expressions

An expression can be classified as one of the following types:

- **Numeric.** A numeric expression returns a numeric value.

Use a numeric expression to perform arithmetic calculations on numeric constants or fields. For example, you can calculate the bonus for each employee by multiplying the current salary by the desired percentage:

```
FIELD BONUS DEFINED BY (CURR_SALE * 0.05) ;
```

For detailed information on numeric expressions, see *Numeric Expressions* on page 5-5.

- **Date.** A date expression returns either a date or an integer that represents the number of days, months, quarters, or years between two dates.

Use a date expression to perform arithmetic calculations that involve dates. For example, you can determine when a customer can expect to receive an order by adding the number of days in transit to the date on which you shipped the order:

```
FIELD DELIVERY USAGE MDY DEFINED BY (SHIPDATE + 5) ;
```

For detailed information on date expressions, see *Date Expressions* on page 5-7.

- **Alphanumeric.** An alphanumeric expression returns an alphanumeric value.

Use an alphanumeric expression to manipulate alphanumeric constants or fields. For example, you can extract the first initial from the FIRST_NAME field with the EDIT function:

```
FIELD FIRST_INIT USAGE A1 DEFINED BY (EDIT (FIRST_NAME, '9$$$$$$$$')) ;
```

For detailed information on alphanumeric expressions, see *Alphanumeric Expressions* on page 5-10.

- **Logical.** A logical expression evaluates to TRUE or FALSE.

Use a logical expression to determine whether a condition is true. For example, you can identify employees in the MIS department who earn more than \$25,000:

```
FIELD MIS25K USAGE I1  
  DEFINED BY ((DEPARTMENT EQ 'MIS') AND (CURR_SALE GT 25000)) ;
```

For detailed information on logical expressions, see *Logical Expressions* on page 5-11.

- **Conditional.** A conditional expression returns one of two or more alternate values based on whether or not a logical condition is true.

Use a conditional expression when the value to be returned depends on the truth of a logical condition. For example, you can give employees a 10% raise if they are in the MIS department, and a 5% raise if they're not:

```
FIELD CURR_SAL DEFINED BY (IF DEPARTMENT EQ 'MIS' THEN CURR_SAL * 1.1
ELSE CURR_SAL * 1.05);
```

For complete information on conditional expressions, see *Conditional Expressions* on page 5-13.

Expressions and Field Formats

When you use an expression to assign a value to a field, you must give the field a format that is consistent with the value returned by the expression. For example, if you create the field FULL_NAME by concatenating the first name and last name, make sure you assign the new field an alphanumeric format as in the following example:

```
FIELD FULL_NAME USAGE A25 DEFINED BY (FIRST_NAME | LAST_NAME);
```

Numeric Expressions

A numeric expression returns a numeric value as a result of performing arithmetic calculations on numeric constants, fields, operators, or functions. When you use a numeric expression to assign a value to a field, that field must have a numeric format. The default format is D12.2.

A numeric expression can consist of:

- A numeric constant. For example:

```
FIELD COUNT USAGE I1 DEFINED BY (1);
```

- Two numeric constants or fields joined by an arithmetic operator. For example:

```
FIELD BONUS DEFINED BY (CURR_SAL * 0.05);
```

For a list of arithmetic operators, see *Arithmetic Operators* on page 5-6.

- A numeric function. For example:

```
FIELD LONGEST_SIDE DEFINED BY (MAX (WIDTH, HEIGHT, DEPTH) );
```

- Two or more numeric expressions joined by an arithmetic operator. For example:

```
FIELD PROFIT DEFINED BY ((RETAIL_PRICE - UNIT_COST) * UNIT_SOLD );
```

Note the use of parentheses to change the order of evaluation of the expression. For information on the order in which Fusion performs numeric operations, see *Order of Evaluation* on page 5-6.

Fusion converts all numeric values to double-precision floating-point format before using them in calculations, and then converts the result back to the specified field format. In some cases, the conversion may result in a rounding difference.

If a number is greater than 10^{75} or less than 10^{-75} , Fusion issues an OVERFLOW or UNDERFLOW warning and displays asterisks for the field value.

Arithmetic Operators

The following is a list of the arithmetic operators you can use in an expression:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

If you divide by zero, Fusion sets the value of the expression to zero.

Order of Evaluation

The order of evaluation can affect the result of an expression. Fusion performs arithmetic operations in the following order:

1. Exponentiation.
2. Division and multiplication.
3. Addition and subtraction.

Operations at the same level are performed from left to right. The order of evaluation can be changed by enclosing expressions in parentheses. You can also use parentheses to improve readability.

Example

Using Parentheses to Alter the Order of Evaluation

The following calculation yields an incorrect result because UNIT_SOLD is first multiplied by UNIT_COST, and then the result is subtracted from RETAIL_PRICE:

```
FIELD PROFIT DEFINED BY ((RETAIL_PRICE - UNIT_COST * UNIT_SOLD) );
```

Using parentheses, the correct result is achieved with the following syntax:

```
FIELD PROFIT DEFINED BY ((RETAIL_PRICE - UNIT_COST) * UNIT_SOLD) ;
```

Date Expressions

A date expression performs arithmetic calculations that involve dates. A date expression returns a date or an integer that represents the number of days, months, quarters, or years between two dates.

A date expression can consist of:

- A calculation that uses arithmetic operators or date functions to return a date. For example:

```
FIELD DELIVERY USAGE MDY DEFINED BY (SHIPDATE + 5) ;
```

- A calculation that uses arithmetic operators or date functions to return an integer that represents the number of days, months, quarters, or years between two dates. For example:

```
FIELD TURNAROUND USAGE I4 DEFINED BY (MDY (ORDERDATE, SHIPDATE)) ;
```

Field Formats for Date Values

Fusion supports date values defined in two ways:

- In a date format such as YMD, MDY, or YYMD. When you use a date format, the value stored internally is an integer that represents the number of days between the date value and the base date (December 31, 1900). When displaying the value, Fusion converts the integer to its corresponding date.
- In integer, packed, or alphanumeric format with date edit options. Fusion treats the value as an integer, a packed decimal, or an alphanumeric string. When displaying the value, Fusion formats it to resemble a date.

The field format affects storage and display. This is illustrated by the following table:

Value	Date Format For example: MDY		Integer, Packed, or Alphanumeric Format For example: A6MDY	
	Stored	Displayed	Stored	Displayed
October 31, 1999	33542	10/31/99	103199	10/31/99
November 01, 1999	33543	11/01/99	110199	11/01/99

Performing Calculations on Dates

The format of a field determines how you can use it in a date expression:

- Calculations on dates in date format can incorporate arithmetic operators and numeric functions.
- Calculations on dates in integer, packed, or alphanumeric format require the use of date functions; using arithmetic operators would return an error message or an incorrect result.

Example

Using Dates in a Calculation

This example illustrates how the field format affects the use of a field in a calculation. The following calculation subtracts the order date, ORDERDATE, from the shipping date, SHIPDATE to calculate the number of days it takes to fill an order:

```
FIELD TURNAROUND USAGE I4 DEFINED BY (SHIPDATE - ORDERDATE) ;
```

An item ordered on October 31, 1999 and shipped on November 1, 1999 should result in a difference of 1 day. The following table shows how the field format affects the result:

	Value in Date Format	Value in Numeric Format
SHIPDATE = November 1, 1999	33543	110199
ORDERDATE = October 31, 1999	33542	103199
TURNAROUND	1	7000

To calculate the correct result using fields in integer, packed, or alphanumeric format, you can use a date function, MDY, which returns the difference between two dates in the form month-day-year. Using the MDY function, you can calculate TURNAROUND as follows:

```
FIELD TURNAROUND USAGE I4 DEFINED BY (MDY (ORDERDATE, SHIPDATE) ) ;
```

For information on date functions, see Appendix B, *Fusion Functions and Subroutines*.

Selecting the Format of the Result Field

A date expression always returns a number. That number may represent a date or the number of days, months, quarters, or years between two dates. When you use a date expression to assign a value to a field, the format you give to the field determines how the result will be displayed.

Consider the following derived fields. The first field declaration calculates how many days it takes to fill an order by subtracting the order date, ORDERDATE, from the shipping date, SHIPDATE. The second calculates a delivery date by adding 5 days to the shipping date:

```
FIELD TURNAROUND USAGE I4 DEFINED BY (SHIPDATE - ORDERDATE) ;
FIELD DELIVERY USAGE MDY DEFINED BY (SHIPDATE + 5) ;
```

In the first field declaration, the date expression returns the number of days it takes to fill an order; therefore, the associated field, **TURNAROUND**, must have an integer format. In the second field declaration, the date expression returns the date on which the item will be delivered; therefore, the associated field, **DELIVERY**, must have a date format.

Working With Dates in Date Format

The following topics describe how to:

- Use a date constant in an expression.
- Extract a date component.
- Combine fields with different components in an expression.

Defining a Date Constant in an Expression

You can define a date constant for an expression. This gives you a starting point for your date calculation. When you use a date constant in a calculation that involves fields in date format, you must enclose the constant in single quotation marks.

Examples

Using a Date Constant in a Calculation

The following example shows how to initialize **STARTDATE** with the date constant 02/28/93:

```
FIELD STARTDATE USAGE MDY DEFINED BY ('022899') ;
```

The next example calculates the number of days elapsed since January 1, 1999:

```
FIELD YEARTODATE USAGE I4 DEFINED BY (CURR_DATE - 'JAN 1 1999') ;
```

Extracting a Date Component

Date components include days, months, quarters, or years. It is easy to extract a component from a field in date format.

You cannot use this technique to extract days, months, or quarters from a date that does not include those components. For example, you cannot extract a month from a date in **YY** format.

Extracting a Date Component From a Field

The following example shows how to extract a month from SHIPDATE, which has the format MDY:

```
FIELD SHIPMONTH USAGE M DEFINED BY (SHIPDATE );
```

If SHIPDATE has the value November 23, 1999, this expression returns the value 11 for SHIPMONTH. Calculations on date components automatically produce valid values for the components. For example, if the current value of SHIPMONTH is 11, the following expression correctly returns the value 2, not 13:

```
FIELD ADDTHREE USAGE M DEFINED BY (SHIPMONTH + 3) ;
```

Combining Fields With Different Components in an Expression

When you use date formats, you can combine fields defined with their components in different orders in the same expression. In addition, you can assign the result of a date expression to a field with a different order of components from the fields in the expression. However, an expression cannot combine dates in date format with dates in integer, packed, or alphanumeric format.

Combining Fields in an Expression

If DATE_PAID has the format YYMD and DUE_DATE has the format MDY, you can combine them in an expression to calculate the number of days that a payment is late:

```
FIELD DAYS_LATE USAGE I4 DEFINED BY (DATE_PAID - DUE_DATE) ;
```

The field DATE_SOLD contains the date on which an item is sold, in YYMD format. You can add 7 days to DATE_SOLD to determine the last date on which the item can be returned, and assign the result to a field with DMY format, as in the following example:

```
FIELD RETURN_BY USAGE DMY DEFINED BY (DATE_SOLD + 7) ;
```

Alphanumeric Expressions

An alphanumeric expression manipulates alphanumeric constants, fields, concatenation operators, and functions to return an alphanumeric value. When you use an alphanumeric expression to assign a value to a field, you must give that field an alphanumeric format.

An alphanumeric expression can consist of:

- A character string enclosed in single quotation marks. For example:

```
FIELD STAT USAGE A2 DEFINED BY ('NY') ;
```

- Two or more alphanumeric fields or constants joined by a concatenation operator. For example:

```
FIELD TITLE USAGE A19 DEFINED BY ('DR.'|LAST_NAME);
```

- An alphanumeric function. For example:

```
FIELD INITIAL USAGE A1 DEFINED BY (EDIT (FIRST_NAME, '9$$$$$$$$'));
```

Concatenating Character Strings

An expression can concatenate two or more alphanumeric constants or fields into a single character string. The concatenation operator takes one of two forms:

Symbol	Represents	Function
	Weak concatenation	Preserves trailing blanks.
	Strong concatenation	Suppresses trailing blanks.

Example

Using Concatenation

The following example shows how to use the EDIT function to both extract the first initial from a first name, and use strong and weak concatenation to produce a character string consisting of the last name, a comma, the first initial, and a period.

Consider the following code:

```
FIELD INIT USAGE A1 DEFINED BY (EDIT (FIRST_NAME, '9$$$$$$$$'));
FIELD NAME USAGE A19 DEFINED BY (LAST_NAME || (',' | INIT | '.'));
```

If FIRST_NAME has the value Chris and LAST_NAME has the value Edwards:

1. The EDIT function extracts the initial C from FIRST_NAME.
2. The expression in parentheses is evaluated, and it returns the following value:
, C.
3. LAST_NAME is concatenated to the string derived in Step 2 to produce

```
Edwards, C.
```

Although LAST_NAME has the format A15, strong concatenation suppresses the trailing blanks.

Logical Expressions

A logical expression evaluates whether a condition is true. There are two kinds of logical expressions, relational and Boolean. A relational expression compares two values (either fields or constants) and returns the value TRUE or FALSE as the result of the comparison. A Boolean expression returns the value TRUE or FALSE as a result of operating on other logical expressions.

You can use a logical expression to assign a value to a numeric field. If the expression is true, the field receives the value 1. If the expression is false, the field receives the value 0.

Relational Expressions

A relational expression returns TRUE or FALSE based on a comparison of two individual values (either fields or constants).

Syntax

How to Compare Numeric Values Using a Relational Expression

`numeric_value1 expression numeric_value2`

where:

`numeric_value1, numeric_value2`

Are numeric constants.

`expression`

Is one of the following values:

- | | |
|-----------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>EQ</code> | Assigns the value TRUE to the expression when <i>numeric_value1</i> is equal to <i>numeric_value2</i> . |
| <code>NE</code> | Assigns the value TRUE to the expression when <i>numeric_value1</i> is not equal to <i>numeric_value2</i> . |
| <code>GE</code> | Assigns the value TRUE to the expression when <i>numeric_value1</i> is greater than or equal to <i>numeric_value2</i> . |
| <code>GT</code> | Assigns the value TRUE to the expression when <i>numeric_value1</i> is greater than <i>numeric_value2</i> . |
| <code>LE</code> | Assigns the value TRUE to the expression when <i>numeric_value1</i> is less than or equal to <i>numeric_value2</i> . |
| <code>LT</code> | Assigns the value TRUE to the expression when <i>numeric_value1</i> is less than <i>numeric_value2</i> . |

Syntax

How to Compare Alphanumeric Values Using a Relational Expression

`alphanumeric_value1 {CONTAINS|OMITS} alphanumeric_value2`

where:

`alphanumeric_value 1, alphanumeric_value2`

Are alphanumeric fields.

CONTAINS

Assigns the value TRUE to the expression when *alphanumeric_value1* contains the character string represented by *alphanumeric_value2*.

OMITS

Assigns the value TRUE to the expression when *alphanumeric_value1* does not contain the character string represented by *alphanumeric_value2*.

You must enclose alphanumeric literals in single quotation marks that contain embedded blanks.

Boolean Expressions

A Boolean expression returns the value TRUE or FALSE as a result of evaluating two logical expressions with an operator.

Syntax

How to Compare Values Using a Boolean Expression

logical_expression1 operator logical_expression2

where:

logical_expression1, logical_expression2

Are logical expressions, enclosed in parentheses.

operator

Is one of the following values:

- | | |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AND | Returns the value TRUE only when <i>logical_expression1</i> and <i>logical_expression2</i> are both TRUE; it returns the value FALSE if either expression is FALSE. |
| OR | Returns the value FALSE only when <i>logical_expression1</i> and <i>logical_expression2</i> are both FALSE; it returns the value TRUE if either expression is TRUE. |
| NOT | Returns the value TRUE if <i>logical_expression2</i> is FALSE; it returns the value FALSE if <i>logical_expression2</i> is TRUE. <i>logical_expression1</i> is not included in this type of expression. |

Conditional Expressions

A conditional expression assigns a value by evaluating one of two alternate expressions. The choice between the two expressions is controlled by the value of a third expression—a logical expression. If the logical expression is TRUE, the first of the alternate expressions is evaluated; if the logical expression is FALSE, the second of the alternate expressions is evaluated. Conditional expressions can include up to 16 IF criteria. The assigned value can be numeric or alphanumeric.

The expressions following THEN and ELSE must result in a format that is compatible with the format assigned to the field. Each of these expressions may itself be a conditional expression, although the expression following IF may not be (for example, IF...IF...). If the expression that follows THEN is a conditional expression, it must be enclosed in parentheses. Parentheses are not necessary after the ELSE expression. Omitting ELSE leaves the value unchanged when the IF condition is not met.

Syntax

How to Create a Conditional Expression

```
IF logical_expression THEN expression1 ELSE expression2
```

where:

```
logical_expression
```

Is a logical expression.

```
expression1
```

Is an expression. It is evaluated if the logical expression is TRUE.

```
expression2
```

Is an expression. It is evaluated if the logical expression is FALSE.

Example

Using a Conditional Expression

The following expression contains a conditional expression that will affect the result of the report request. The DEFINE is contained in the Master File:

```
FIELD BANK_NAME USAGE A20 DEFINED BY (IF BANK_NAME EQ ' ' THEN 'NONE' ELSE  
BANK_NAME) ;
```

The request

```
TABLE FILE EMPLOYEE  
PRINT CURR_SAL AND BANK_NAME  
BY EMP_ID BY BANK_ACCT  
END
```

produces the following report:

PAGE 1

EMP_ID	BANK_ACCT	CURR_SAL	BANK_NAME
-----	-----	-----	----
071382660		\$11,000.00	NONE
112847612		\$13,200.00	NONE
117593129	40950036	\$18,480.00	STATE
119265415		\$9,500.00	NONE
119329144	160633	\$29,700.00	BEST BANK
123764317	819000702	\$26,862.00	ASSOCIATED
126724188		\$21,120.00	NONE
219984371		\$18,480.00	NONE
326179357	122850108	\$21,780.00	ASSOCIATED
451123478	136500120	\$16,100.00	ASSOCIATED
543729165		\$9,000.00	NONE
818692173	163800144	\$27,062.00	BANK ASSOCIATION

END OF REPORT

Example

Using a Conditional Expression in an IF Test

You can define a logical condition and then refer to its value as TRUE or FALSE in a screening (IF) test. This technique eliminates the IF-THEN-ELSE expression; the IF test in the report request selects the appropriate records based on the value of the MYTEST field. The DEFINE is contained in the Master File.

The request

```
FIELD MYTEST DEFINED BY ((CURR_SAL GE 11000) OR (DEPARTMENT EQ 'MIS'));  
TABLE FILE EMPLOYEE  
PRINT CURR_SAL AND DEPARTMENT  
BY EMP_ID  
IF MYTEST IS TRUE  
END
```

produces the following report:

PAGE 1

EMP_ID	CURR_SAL	DEPARTMENT
-----	-----	-----
071382660	\$11,000.00	PRODUCTION
112847612	\$13,200.00	MIS
117593129	\$18,480.00	MIS
119329144	\$29,700.00	PRODUCTION
123764317	\$26,862.00	PRODUCTION
126724188	\$21,120.00	PRODUCTION
219984371	\$18,480.00	MIS
326179357	\$21,780.00	MIS
451123478	\$16,100.00	PRODUCTION
543729165	\$9,000.00	MIS
818692173	\$27,062.00	MIS

END OF REPORT

CHAPTER 6

Fusion Security

Topics:

- Implementing Database Security
- Stored Procedure Security

Security rules can be defined in any Fusion Master File to ensure that users have access only to those resources for which they have explicit authorization.

Fusion access control mechanisms complement and extend other security rules. Fusion respects restrictions previously defined by an underlying security system, such as RACF™, CA-ACF2®, CA-TOP SECRET®, or UNIX® Security; these packages typically work at the file or table level. Fusion also permits application programs to supply passwords for row or field-level security checks. You can invoke additional features in a stored procedure.

Fusion access control is provided on a file-by-file basis by including security attributes in the Fusion Master File. For a complete discussion of Fusion Master Files, see Chapter 2, *Understanding Your Fusion Schema*.

Implementing Database Security

A Fusion Master File includes an optional security section that controls access to the data source it describes. If it exists, the security section must begin with a DBA attribute. The DBA attribute must immediately follow the END command that marks the end of the schema declarations.

The security section can include:

- The name or password of the user authorized to access a file.
- The type of access the user is granted.
- The classes, fields, or ranges of data values to which the user's access is restricted.

If the Fusion Master File does not have a security section, users have unlimited access to the data source. The Database Administrator (DBA) can encrypt the Fusion Master File to prevent unauthorized viewing of the security information.

Reference

Security Attributes in a Fusion Master File

The security section can consist of the following attributes:

Attribute	Purpose
DBA	Identifies the database administrator.
USER PASS	Identifies the password needed to access a data source.
ACCESS	Identifies the type of access permitted for the password.
RESTRICT	Places field, class, or field value restrictions on a password.
NAME	Used with RESTRICT. Names the restricted fields or classes.
VALUE	Used with RESTRICT. Supplies a test expression used to determine access to the class.
DBAFILE	Identifies a central file that contains the security attributes for the data source.

Example**Including a Security Section in a Fusion Master File**

The following Fusion Master File includes a security section. It is found below the END command:

```

FILENAME  BUILDING  FILETYPE  FUSION
CLASS    BUILDING  KEYS  S1
FIELD    BUILDING_ID  FORMAT  A8  INDEX  ON
FIELD    ADDRESS      FORMAT  A48
FIELD    STATE        FORMAT  A16
FIELD    CITY         FORMAT  A16
FIELD    ZIP          FORMAT  A10
END
DBA = FRED, $
USER = BILL, ACCESS = RW, $

```

Identifying the Database Administrator

Use the DBA attribute to supply the password for the Database Administrator (DBA). The DBA attribute must be the first attribute in the security section. Only a user who supplies the DBA password can change the security attributes in the Fusion Master File.

The Database Administrator has unlimited access to a file and all joined files. Therefore, you cannot specify restrictions that apply to the DBA attribute. The Database Administrator has the freedom to change any of the access control attributes. If you change the DBA password in the Fusion Master File, you must use the RESTRICT command in a stored procedure to inform Fusion that this change has been made. Only users with the DBA password may use the RESTRICT command. Unless the RESTRICT command is issued, Fusion will assume this is an attempt to bypass the restriction rules.

You cannot encrypt and decrypt Fusion Master Files or restrict existing data sources without setting the correct DBA password in the procedure using the encryption commands. You can have different DBA passwords for each data source.

Syntax**How to Identify the Database Administrator**

```
DBA = dbapass, $
```

where:

dbapass

Is a one- to eight-character password for the Database Administrator.

Procedure**How to Inform Fusion of a Change to the DBA Password**

To change the DBA password :

1. Update the DBA password in the Master Files. See *How to Identify the Database Administrator* on page 6-3 for more information.
2. Issue the SET PASS command using the old DBA password to indicate that you have authority to change the security rules:

```
SET PASS = oldpassword
```

3. Issue the RESTRICT command to write the new DBA password to the data source:

```
RESTRICT  
master_file  
master_file  
.  
.  
END
```

4. Issue the SET PASS command using the new password to indicate that you have DBA authority:

```
SET PASS = newpassword
```

Note: Each command must be issued on a separate line followed by enter key.

Example**Informing Fusion of a Change to the DBA Password**

This example changes the DBA password from ABC to XYZ in two separate Master Files, carfusx and carf.

1. The following commands are issued after the DBA password has been changed in the Master Files:

```
>>SET PASS = ABC  
>>RESTRICT  
>carfusx
```

The following displays:

```
File /home/carfusx.fus updated.
```

2. When the name of the second file is entered,

```
>carf
```

the three partitions are updated, and the following is displayed:

```
File /home/carf.fus updated.  
File /home/carf2.fus updated.  
File /home/carf3.fus updated.
```

- The following command is issued:

```
>END
```

- The following command indicates that you have DBA authority:

```
>>SET PASS=XYZ
```

Note: The file being updated is platform specific. MVS will display a data set name when updated, while UNIX and NT will display full file paths.

Identifying a User

The USER attribute supplies an arbitrary code name that identifies a user who is authorized to access a file. You cannot specify a USER attribute alone; you must follow it with at least an ACCESS restriction.

Syntax

How to Identify a User

```
{USER|PASS} = username,
```

where:

username

Is a 1 to 8-character code name.

Example

Identifying a User With the USER Attribute

The following USER attribute identifies the code name TOM:

```
USER = TOM,
```

Querying User Identity

Fusion maintains a list of the files for which a user has set specific passwords. To see the list of files, issue a query in a stored procedure.

Syntax

How to Query a List of Passwords

```
? PASS
```

Identifying a User for a Specific File

Before using a secured file, a user must enter a password using the SET PASS or SET USER command in a stored procedure. A single user may have different passwords in different files. They may issue their passwords at any time before using the file and can issue a different password afterward to access another file.

Syntax

How to Establish User Identity for a Specific File

```
SET {PASS|USER} = name [ [IN {file/*} [NOCLEAR] ] , name [IN file] ...]
```

where:

name

Is the user's name or password.

file

Is the file name that the password applies to.

*

Indicates that *name* replaces all passwords active in all files. When * is used when setting a password, the list of active passwords collapses to one entry with no associated file name. To retain the file name list, use the NOCLEAR option.

NOCLEAR

Replaces all passwords in the list of active passwords while retaining the list of files that have password restrictions.

If the password entered is not valid or is inadequate for the type of access requested, the user is denied access, and the following message displays:

```
(FOC047) THE USER DOES NOT HAVE SUFFICIENT ACCESS RIGHTS TO THE FILE:
filename
```

Example

Assigning an Identity to a User

In the following example, the password TOM is in effect for all files that do not have a specific password designated for them:

```
SET PASS=TOM
```

For the next example, in file ONE the password is BILL, and in file TWO the password is LARRY. No other files have passwords set for them:

```
SET PASS=BILL IN ONE, LARRY IN TWO
```

Here, all files have the password SALLY except files SIX and SEVEN, which have the 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
```

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
```

Specifying the Type of Access for a User

The ACCESS attribute defines the type of access granted to a user, and determines what a user can do to a file. Every user, except the DBA, must have an ACCESS attribute.

An ACCESS attribute determines what a user can do to a file. Every user must be assigned an ACCESS attribute.

Syntax

How to Specify the Access Type

```
ACCESS = attribute
```

where:

attribute

Determines the type of access a user has to a file. The options are:

- | | |
|----|---------------------------------------------------------|
| R | Allows the user to read a file. |
| W | Permits the user to write to a file. |
| RW | Allows the user to read a file and write new instances. |
| U | Allows the user to update a file. |

Example

Specifying Access

The following is a complete access control declaration, consisting of a USER and ACCESS attribute:

```
USER = TOM, ACCESS = RW,$
```

This declaration gives the user Tom read and write access to the file.

Limiting a User's Access Within a File

The RESTRICT attribute limits access to specific fields, values, or classes within a file. The RESTRICT attribute is optional; without it, the user has unlimited access to a file.

Syntax

How to Limit Access Within a File

```
RESTRICT= type ,NAME= {name|SYSTEM} [,VALUE = test ],$
```

where:

type

Specifies the restrictions a user has for a specific file. The options are:

<code>FIELD</code>	Prevents the user from accessing the field specified in the NAME attribute.
<code>SEGMENT</code>	Prevents the user from accessing the class specified in the NAME attribute.
<code>PROGRAM</code>	Indicates that the external program specified in the NAME attribute must be called to verify access. This option may not be available on all platforms.
<code>SAME</code>	Imposes the same restrictions as the user specified in the NAME attribute. No more than four nested SAME users are valid.
<code>NOPRINT</code>	Allows the user to retrieve, but not display, the field specified in the NAME attribute. Instead, blanks display for alphanumeric fields, and zeros for numeric fields. This restriction allows users with different access rights to issue the same request.
<code>VALUE</code>	Prevents the user from accessing field values that make the test expression false. The NAME attribute specified the class on which to perform the test.

name

Is the name of the field or class to restrict. When used after NOPRINT, must be a field name. Any attempt to access a field or class that is the value of the NAME attribute is rejected as beyond the user's access rights.

`SYSTEM`

Restricts every class in the file, including descendant classes. This value can only be used with value tests.

test

Is the value test that the data must satisfy before the user can have access.

Reference

Using RESTRICT

You can use RESTRICT in the following ways:

- Restrict any number of classes and fields by providing multiple RESTRICT attributes. For example, the following declaration restricts Harry from using both field STAFFLEVEL and class OTHRSTFF:

```
USER = HARRY, ACCESS = R, RESTRICT = FIELD, NAME =
STAFFLEVEL, $
RESTRICT = CLASS, NAME = OTHRSTFF, $
```

- Use a common set of restrictions for more than one password by issuing the RESTRICT=SAME command. This command requires you to provide an existing user name. The new user will be subject to the same restrictions as the named user. You can then add additional restrictions as needed. In the following example, both Sally and Harry have the same access privileges as BILL. In addition, Sally is not allowed to read the SALARY field.

```
USER = BILL, ACCESS = R, RESTRICT = FIELD, NAME = PROJECTS, $
RESTRICT = FIELD, NAME = LOCATION, $
USER = SALLY, ACCESS = R, RESTRICT = SAME, NAME = BILL, $
RESTRICT = FIELD, NAME = SALARY, $
USER = HARRY, ACCESS = R, RESTRICT = SAME, NAME = BILL, $
```

You can restrict the values a user can access by providing a test condition with the RESTRICT attribute. The user will only have access to those values that satisfy the test condition. For example:

```
USER = BILL, ACCESS = R, RESTRICT = VALUE, NAME = PROJECTS,
VALUE = PROJTYPE NE 'SPECIAL' AND LOCATION EQ 'NY', $
```

Note: You can restrict the values the user can read from the file and the values the user can write to a file. You must impose each of these restrictions separately; one does not imply the other.

Example

Limiting Access Within a File

In the following example, Bill has read-only access to everything in the file except the PROJECTS class:

```
USER = BILL, ACCESS = R, RESTRICT = SEGMENT, NAME = PROJECTS, $
```

Combining RESTRICT and ACCESS Attributes to Limit Access

The ACCESS and RESTRICT attributes can be used in the following ways to limit access:

- Set ACCESS to R and RESTRICT to VALUE to limit the values a user can read. This type of restriction prevents the user from seeing any data values other than those that meet a test condition. A RESTRICT = VALUE attribute with ACCESS = R acts as a WHERE condition in a request. Therefore, the syntax for ACCESS = R value restrictions must follow the rules for a WHERE test in a report request.
- Restrict the values a user can write to a file by setting ACCESS to W and RESTRICT to VALUE. You can also use ACCESS = W and RESTRICT = VALUE to limit the data values in the file for which a user can provide new values. With ACCESS = W, the user will be able to access all data values in the file; however, the user will be prohibited from entering certain values or new values for certain existing fields.
- To prevent a user both from entering certain values and from seeing other values, issue two RESTRICT declarations: one with ACCESS set to W, which limits the values a user can write or alter, and one with ACCESS set to R, which limits the values the user can see. ACCESS = RW is meaningless with a RESTRICT = VALUE declaration.

Note: Should several fields have tests performed on them, separate VALUE commands must be provided. Each test must name the class to which it applies. If a single test condition exceeds the allowed length of a line, it can be provided in sections. Each section must start with the VALUE command.

Example

Using ACCESS and RESTRICT Attributes

With the following restriction, Tony can only see records from the western division:

```
USER = TONY  ACCESS = R  RESTRICT = VALUE,  NAME = IDCLASS,
          VALUE = DIVISION EQ 'WEST', $
```

The following example illustrates a situation where several fields have tests performed on them:

```
USER = RAY,  ACCESS = R,  RESTRICT = VALUE,  NAME = IDCLASS,
          VALUE = DIVISION EQ 'EAST' OR 'WEST', $
                                NAME = IDCLASS,
          VALUE = SALARY LE 10000, $
```

The following example provides the test condition in sections:

```
USER = SAM,  ACCESS = R,  RESTRICT = VALUE,  NAME = IDCLASS,
          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 OR command.

Example**Applying a Test Condition to a Parent Class**

You can apply the test conditions to the parent classes of the data classes on which the tests are applicable. Consider the following example:

```
USER = RAY, ACCESS = R, RESTRICT = VALUE, NAME = IDCLASS,
      VALUE = DIVISION EQ 'EAST' OR 'WEST', $
                                NAME IDCLASS,
      VALUE = SALARY LE 10000, $
```

The field named SALARY is actually part of a class named COMPCLAS. Since the test specifies NAME = IDCLASS, the test is effective for requests on its parent, IDCLASS. In this case, the request to 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 class of IDCLASS. If, however, the test was made effective on COMPCLAS, that is, NAME = COMPCLAS, then the full name of everyone in the file could be retrieved, but with the salary information of only those meeting the test condition.

Storing Security Information in a Central File

You can store security information for many Fusion Master Files in one central file. Doing this simplifies password administration. Passwords that are applicable to a group of data sources only have to be stored once. This gives you the ability to join files with different DBA passwords. Individual DBA information remains in effect for each file in a join. Therefore, you can distribute access rights for one application to the control of a different DBA by assigning a password in your system.

The central DBAFILE is a standard Fusion Master File with a security section. Each individual Fusion Master File can then point to this central control file with the DBAFILE attribute. The central DBAFILE and all of its participating Fusion Master Files must share the same DBA password. This prevents individuals from substituting their own security attributes. All of these Fusion Master Files should be encrypted.

You can define passwords and restrictions that apply to every Fusion Master File that points to the DBAFILE. You can also identify passwords and restrictions for specific Fusion Master Files by including a FILENAME attribute in the DBAFILE. File-specific security information begins with a FILENAME attribute (for example, FILENAME = TWO). File-specific restrictions override general restrictions. In case of 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 in a FILENAME section for a particular file.

The security section of the DBAFILE must start with a DBA attribute, after which it can contain a list of passwords and restrictions. These passwords apply to all files that reference this DBAFILE. Value restrictions accumulate; all value restrictions must be satisfied before retrieval.

Syntax

How to Point to a Central File From a Master File

Include the following after the END command

```
DBA = dbaname, DBAFILE = dbafilename, $
```

where:

dbaname

Is the DBA password in the central file.

dbafilename

Is the name of the central file.

Example

Using a Common DBAFILE

The following example shows a group of Fusion Master Files that share a common DBAFILE. Master Files ONE, TWO, and THREE share the same DBAFILE, FOUR:

Master File ONE:

```
ONE MASTER  
FILENAME ONE  
.  
.  
.  
END  
DBA = ABC, DBAFILE = FOUR, $
```

Master File TWO:

```
TWO MASTER  
FILENAME TWO  
.  
.  
.  
END  
DBA = ABC, DBAFILE = FOUR, $
```

Master File THREE:

```
THREE MASTER
FILENAME THREE
.
.
.
END
DBA = ABC,
DBAFILE = FOUR,$
```

Master File FOUR, the central DBAFILE:

```
FOUR MASTER
FILENAME FOUR
SEGNAME mmmmm
FIELDNAME fffff
END
DBA = ABC,$
    PASS = BILL, ACCESS = R,$
    PASS = JOE, ACCESS = R,$
FILENAME = TWO,$
    PASS = HARRY, ACCESS = RW,$
FILENAME = THREE,$
    PASS = JOE, ACCESS = R, RESTRICT ..., $
    PASS = TOM, ACCESS = R,$
```

Renaming a Master File

It is possible, although not recommended, to give a Fusion Master File a name that is different from its FILENAME attribute; for example, Fusion Master File ONE can include the attribute FILENAME XONE.

The FILENAME attribute used in the central DBAFILE to assign security restrictions for an individual Master File must be identical to the FILENAME attribute found in the individual Fusion Master File. This rule prevents users from renaming a Fusion Master File to a name not known by the DBAFILE.

Example

Assigning Security Restrictions to a Renamed Master File

To assign security restrictions to Master File ONE, the central DBAFILE must use the attribute FILENAME = XONE:

```
ONE MASTER
FILENAME XONE          -* FILENAME not the same as Fusion Master File
name.
.
.
.
END
DBA = ABC, DBAFILE = FOUR,$      -* Pointer to DBAFILE

FOUR MASTER          -* This is the DBAFILE
FILENAME FOUR
.
.
.
END
DBA = ABC,$
.
.
.
FILENAME = XONE,$      -* Beginning of restrictions for Fusion Master File
ONE
.
.
.
```

Encrypting a Master File

Since the restriction information for a data source is stored in its Fusion Master File, encrypt the Fusion Master File to prevent users from examining the restriction rules. Only the Database Administrator can encrypt a Fusion Master File. 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.

The process can be reversed if you wish to change the restrictions. The command to restore the Fusion Master File to a readable form is `DECRYPT`. The DBA password must be issued with the `SET` command before the file can be decrypted.

Note: You must issue these commands in a stored procedure.

Example

Encrypting and Decrypting a Master File

The following is an example of the complete procedure:

```
SET PASS=JONES76
ENCRYPT FILE PERS
```

This procedure is decrypted with the following:

```
SET PASS=JONES76
DECRYPT FILE PERS
```

Stored Procedure Security

Most data security issues are best handled by the Fusion access control facility. However, some additional data security facilities are incorporated in Dialogue Manager. These are:

- Setting passwords in encrypted stored procedures.
- Defining variable passwords.
- Encrypting and decrypting stored procedures.
- Locking stored procedure users out of Fusion.

Setting Passwords in Encrypted Stored Procedures

Passwords can be set in stored procedures and tied to different portions of stored procedures.

The user does not need to issue the password with the `SET` command. The password is not visible to anyone, and the procedure must be encrypted so that the password cannot be revealed by printing the procedure. For more information, see the *EDA Stored Procedures Reference Manual*.

Syntax

How to Set a Password in an Encrypted Stored Procedure

```
-PASS password
```

where:

```
password
```

Is an encrypted password in a stored procedure.

Defining Variable Passwords

The Dialog Manager command `-PASS` can have a variable attached to it.

This command is only visible when you edit the stored procedure. It does not appear when the `ECHO` option is `ALL` and is not printed in a batch run log. Note that data with the `ECHO` option will be returned as messages to the EDA client.

How to Define a Variable Password

```
-PASS &variable
```

where:

```
variable
```

Is a variable password.

Example

Assigning a Variable Password

For example:

```
-PASS &MYPASS
```

Viewing an Encrypted Stored Procedure

You can keep the actual text of a stored procedure confidential while allowing users to execute the stored procedure. You may want to do this because there is confidential information in the stored procedure or because you do not want the stored procedure changed by unauthorized users. You can protect a stored procedure from unauthorized users with the `ENCRYPT` command. Any user can execute an encrypted stored procedure, but you must decrypt the stored procedure to view it. Only a user with the DBA password can decrypt the stored procedure.

Example

Encrypting and Decrypting a Stored Procedure

You use the following to encrypt the stored procedure named SALERPT:

```
SET PASS = DOHIDE  
ENCRYPT FILE SALERPT FOCEXEC
```

The stored procedure can only be viewed by decrypting it, as follows:

```
SET PASS = DOHIDE  
DECRYPT FILE SALERPT FOCEXEC
```

Note: You must issue these commands in a stored procedure.

CHAPTER 7

The Fusion Catalog

Topics:

- The Fusion Catalog Master File
- Querying the Fusion Catalog Master File

A catalog is a repository that the system itself can use to obtain detailed information about its own objects. The Fusion Catalog provides information about Master Files, Access Files, data source locations, and index information, and is available to the end user. The Fusion Catalog utilizes the FUSCAT Master File that allows you to query your Fusion Master Files and Fusion Access Files.

The Fusion Catalog Master File

The FUSCAT Master File provides information about:

- Available Master Files and Access Files.
- File attributes.
- Classes, fields, and index attributes.
- Join information.
- Horizontal, vertical, and MDI partitions.

The FUSCAT Master File also provides information about any command allowed in a Fusion Master File or Fusion Access File. The FUSCAT Master File contains all the acceptable Fusion Master File and Fusion Access File commands, properly arranged in different classes. You can issue any query about Fusion Master Files and Fusion Access Files as a TABLE request against FUSCAT.

The FUSCAT Master File describes other Master Files and Access Files. The difference between this Master File and any other Master File is the value of its FILETYPE attribute. Other Master Files have the file type FUSION, which instructs it to retrieve data from a Fusion data source. The FUSCAT Master File has the file type FCT, which instructs it to retrieve information from a Master File or Access File.

Describing the Fusion Catalog Master File

The description of the FUSCAT Master File follows:

```

FILENAME FUSCAT FILETYPE FCT

- * FILE ATTRIBUTE SPECIFICATION
CLASSNAME FILSPEC
  FIELDNAME FULLNAME          ALIAS FULLNAME          FORMAT A80      ACTUAL A80
  HELPMESSAGE 'FULLY QUALIFIED MASTER NAME'
  FIELDNAME MAS_DATE_MODIFIED ALIAS DATE              FORMAT YYMD     ACTUAL A8
  FIELDNAME MAS_TIME_MODIFIED ALIAS TIME              FORMAT A8       ACTUAL A8
  HELPMESSAGE 'HH:MM:SS'
  FIELDNAME MAS_SIZE          ALIAS SIZE              FORMAT I9       ACTUAL I4
  HELPMESSAGE 'SIZE IN BYTES'
  FIELDNAME FILENAME          ALIAS MASTERNAME       FORMAT A8       ACTUAL A8
  FIELDNAME FILETYPE          ALIAS SUFFIX           FORMAT A8       ACTUAL A8
  ACCEPT FIX OR COM OR ADBSIN OR DATACOM OR IDMSR OR SQLDS
  OR FOC OR FUS OR IMS OR ISAM OR CPMILL
  OR M204IN OR SQLDS OR SQLORA OR SUPRA OR S2K
  OR SQLDBC OR TOTIN OR VSAM
  FIELDNAME ACCESSFILE        ALIAS ACCESSFILE       FORMAT A8       ACTUAL A8
  FIELDNAME ACCFULLNAME       ALIAS ACCFULLNAME      FORMAT A80      ACTUAL A80
  FIELDNAME ACC_DATE_MODIFIED ALIAS DATE              FORMAT YYMD     ACTUAL A8
  FIELDNAME ACC_TIME_MODIFIED ALIAS TIME              FORMAT A8       ACTUAL A8
  HELPMESSAGE 'HH:MM:SS'
  FIELDNAME ACC_SIZE          ALIAS SIZE              FORMAT I9       ACTUAL I4
  HELPMESSAGE 'SIZE IN BYTES'
  FIELDNAME REMARKS           ALIAS REMARKS          FORMAT A80      ACTUAL A80
  FIELDNAME FDFCENT           ALIAS FDFC             FORMAT A2       ACTUAL A2
  FIELDNAME FYRTHRESH         ALIAS FYRT             FORMAT A2       ACTUAL A2

```

```

-* CLASS ATTRIBUTE SPECIFICATION
CLASSNAME SEGSPEC PART_OF FILSPEC KEYS S1
FIELDNAME CLASSECT          ALIAS SEGNAME      FORMAT A8      ACTUAL A8
FIELDNAME ENCRYPT            ALIAS ENCRYPT      FORMAT A7      ACTUAL A7
                        ACCEPT ENCRYPT OR '
FIELDNAME LOCATION          ALIAS SEGLOCATION  FORMAT A8      ACTUAL A8
FIELDNAME SECTION          ALIAS SECTION   FORMAT A8      ACTUAL A8
FIELDNAME PART_OF          ALIAS PARENT    FORMAT A8      ACTUAL A8
FIELDNAME KEYS              ALIAS STANY     FORMAT A3      ACTUAL A3
DEFINE CLASSNAME/A8=(IF SECTION EQ ' ' THEN CLASSECT);

-* CLASS TARGET SPECIFICATION
CLASSNAME TGTSPEC PART_OF SEGSPEC KEYS S1
FIELDNAME TARGET_OF        ALIAS TARGETOF   FORMAT A8      ACTUAL A8

CLASSNAME SUB_OF SUBCLASS_OF SEGSPEC
FIELDNAME SUBCLASS_OF      ALIAS STU         FORMAT A8      ACTUAL A8

-* FIELD ATTRIBUTE SPECIFICATION
CLASSNAME FLDSPEC PART_OF SEGSPEC KEYS S0
FIELDNAME FIELDNAME        ALIAS FIELD      FORMAT A66     ACTUAL A66
FIELDNAME ALIAS            ALIAS ALIAS      FORMAT A66     ACTUAL A66
FIELDNAME FORMAT          ALIAS FORMAT      FORMAT A8      ACTUAL A8
FIELDNAME ACTUAL          ALIAS ACTUAL      FORMAT A8      ACTUAL A8
FIELDNAME DEFCENT         ALIAS DFC        FORMAT A2      ACTUAL A2
FIELDNAME YRTHRESH        ALIAS YRT        FORMAT A2      ACTUAL A2
FIELDNAME TITLE           ALIAS TITLE      FORMAT A64     ACTUAL A64
FIELDNAME HELPMESSAGE     ALIAS HELP      FORMAT A78     ACTUAL A78
FIELDNAME DESCRIPTION     ALIAS DESC      FORMAT A43     ACTUAL A43
FIELDNAME MISSING         ALIAS MISSING   FORMAT A7      ACTUAL A7
                        ACCEPT MISSING OR NULLS OR '
FIELDNAME MAXVALUES       ALIAS MAXVAL    FORMAT I9      ACTUAL I4
FIELDNAME INDEX_ON        ALIAS ON          FORMAT A2      ACTUAL A2
FIELDNAME INDEXLOCATION    ALIAS INDEXLOCATION  FORMAT A8      ACTUAL A8
FIELDNAME OLAP_WITHIN    ALIAS FLDWITHIN  FORMAT A66     ACTUAL A66

-* HORIZONTAL PARTITIONS
CLASSNAME ACCDATA PART_OF FILSPEC KEYS S0
FIELDNAME HORIZONTAL_PARTITION_FILE
                        ALIAS PARTITION      FORMAT A80     ACTUAL A80
FIELDNAME HOR_PART_DATE_MODIFIED
                        ALIAS DATE           FORMAT YYMD    ACTUAL A8
FIELDNAME HOR_PART_TIME_MODIFIED
                        ALIAS TIME           FORMAT A8      ACTUAL A8
                        HELPMESSAGE 'HH:MM:SS'
FIELDNAME HOR_PART_SIZE   ALIAS SIZE           FORMAT I9      ACTUAL I4
                        HELPMESSAGE 'SIZE IN BYTES'
FIELDNAME PART_EXIST     ALIAS FILE_EXIST    FORMAT A1      ACTUAL A1
                        ACCEPT 'Y' OR 'N'
DEFINE D_PARTITION/A85=(IF PART_EXIST EQ 'N' THEN
                        '(?) '|PARTITION
                        ELSE PARTITION);

CLASSNAME WHLINES PART_OF ACCDATA KEYS S0
FIELDNAME WHERE           ALIAS WHERECLAUSE  FORMAT A80     ACTUAL A80

-* PHYSICAL NAME FOR CLASS LOCATION
CLASSNAME SEGLOC PART_OF SEGSPEC KEYS S0
FIELDNAME LOCATION_CLASS_FILE ALIAS SEGDATA    FORMAT A80     ACTUAL A80
FIELDNAME LOC_CLASS_DATE_MODIFIED
                        ALIAS DATE           FORMAT YYMD    ACTUAL A8
FIELDNAME LOC_CLASS_TIME_MODIFIED
                        ALIAS TIME           FORMAT A8      ACTUAL A8
                        HELPMESSAGE 'HH:MM:SS'
FIELDNAME LOC_CLASS_SIZE  ALIAS SIZE           FORMAT I9      ACTUAL I4
                        HELPMESSAGE 'SIZE IN BYTES'
FIELDNAME SEG_EXIST      ALIAS FILE_EXIST    FORMAT A1      ACTUAL A1
                        ACCEPT 'Y' OR 'N'
DEFINE D_SEGDATA/A85=(IF SEG_EXIST EQ 'N' THEN
                        '(?) '|SEGDATA
                        ELSE SEGDATA);

```

```

-* PHYSICAL NAME FOR FIELD INDEX
CLASSNAME IDXLOC PART_OF FLDSPEC KEYS S0
FIELDNAME INDEX_LOCATION_FILE ALIAS IDXDATA FORMAT A80 ACTUAL A80
FIELDNAME IDX_CLASS_DATE_MODIFIED
ALIAS DATE FORMAT YYMD ACTUAL A8
FIELDNAME IDX_CLASS_TIME_MODIFIED
ALIAS TIME FORMAT A8 ACTUAL A8
HELPMESSAGE 'HH:MM:SS'
FIELDNAME IDX_CLASS_SIZE
ALIAS SIZE FORMAT I9 ACTUAL I4
HELPMESSAGE 'SIZE IN BYTES'
FIELDNAME IDX_EXIST ALIAS FILE_EXIST FORMAT A1 ACTUAL A1
ACCEPT 'Y' OR 'N'
DEFINE D_IDXDATA/A85=(IF IDX_EXIST EQ 'N' THEN
' (?) ' |IDXDATA
ELSE IDXDATA);

-* PHYSICAL NAME FOR MDI
CLASSNAME LOCMDI PART_OF TGTSPEC KEYS S0
FIELDNAME MDI_LOCATION_FILE ALIAS MDIDATA FORMAT A80 ACTUAL A80
FIELDNAME MDI_CLASS_DATE_MODIFIED
ALIAS DATE FORMAT YYMD ACTUAL A8
FIELDNAME MDI_CLASS_TIME_MODIFIED
ALIAS TIME FORMAT A8 ACTUAL A8
HELPMESSAGE 'HH:MM:SS'
FIELDNAME MDI_CLASS_SIZE
ALIAS SIZE FORMAT I9 ACTUAL I4
FIELDNAME MDI_EXIST ALIAS FILE_EXIST FORMAT A1 ACTUAL A1
ACCEPT 'Y' OR 'N'
DEFINE D_MDIDATA/A85=(IF MDI_EXIST EQ 'N' THEN
' (?) ' |MDIDATA
ELSE MDIDATA);

-*****

-* ACCEPT KEYWORD SPECIFICATION: CHARACTER STRINGS
CLASSNAME ACCEPTST PART_OF FLDSPEC
FIELDNAME ACCEPTST ALIAS ACCEPTSTR FORMAT A20 ACTUAL A20
HELPMESSAGE 'USED TO COMPUTE ACCEPT_STRING'
FIELDNAME ACCEPTID ALIAS ACCEPTIDN FORMAT A20 ACTUAL A20
HELPMESSAGE 'USED TO COMPUTE ACCEPT_STRING'
DEFINE ACCEPT_STRING/A20=(IF ACCEPTST NE ' ' THEN ACCEPTST ELSE ACCEPTID);

-* ACCEPT KEYWORD SPECIFICATION: NUMBERS
CLASSNAME ACCEPTIN PART_OF FLDSPEC
FIELDNAME ACCEPT_NUMBERS ALIAS ACCEPTINT FORMAT I9 ACTUAL I4

-* ACCEPT KEYWORD SPECIFICATION: ACCEPT WITH RANGE
CLASSNAME ACCEPTRG PART_OF FLDSPEC
FIELDNAME ACCEPT_RANGE ALIAS ACCEPTINT FORMAT I9 ACTUAL I4

-* DEFINE KEYWORD SPECIFICATION
CLASSNAME CLDFSPEC PART_OF SEGSPEC
FIELDNAME DEFINE_TEXT_NEW_STYLE_MASTER
ALIAS DEFINETEXT FORMAT A20 ACTUAL A20

-* DEFINE KEYWORD SPECIFICATION
CLASSNAME DEFSPEC PART_OF SEGSPEC
FIELDNAME DEFINE_TEXT_OLD_STYLE_MASTER
ALIAS DEFTXT FORMAT A20 ACTUAL A20

-* GROUP ATTRIBUTE SPECIFICATION
CLASSNAME GRPSPEC PART_OF SEGSPEC
FIELDNAME GROUPNAME ALIAS FIELD FORMAT A66 ACTUAL A66
FIELDNAME ALIAS ALIAS ALIAS FORMAT A66 ACTUAL A66
FIELDNAME USAGE ALIAS FORMAT FORMAT A8 ACTUAL A8
FIELDNAME INDEX ALIAS ON FORMAT A2 ACTUAL A2

-* INDEX GROUP'S ATTRIBUTE SPECIFICATION
CLASSNAME GDXSPEC PART_OF GRPSPEC KEYS S1
FIELDNAME MDI_LOCATION_FOR_GROUP_FIELD
ALIAS MDLOCATION FORMAT A8 ACTUAL A8

```

```

-* MDI INDEX FIELD'S ATTRIBUTE SPECIFICATION
CLASSNAME IDXSPEC PART_OF FLDSPEC KEYS S1
  FIELDNAME MDI_LOCATION_FOR_FIELD
      ALIAS MDLOCATION  FORMAT A8      ACTUAL A8
      ALIAS MAXVAL    FORMAT I9      ACTUAL I4
  FIELDNAME WITHIN_MAXVALUES
      ALIAS MDIWITHIN  FORMAT A66    ACTUAL A66

-* JOIN ATTRIBUTE SPECIFICATION:

-* ** STATIC JOIN
  CLASSNAME JOINST PART_OF FLDSPEC KEYS S0
    FIELDNAME JOINFILE      ALIAS FQUALNAME  FORMAT A8
    FIELDNAME JOINCLASS     ALIAS FCLASSNAME  FORMAT A12
    FIELDNAME JOINFIELD     ALIAS FFIELDNAME  FORMAT A66
    DEFINE RJOIN/A86 = JOINFILE | JOINCLASS | JOINFIELD;
    FIELDNAME ST_ONLY       ALIAS LINKONLY    FORMAT A4      ACTUAL A4
      ACCEPT ONLY OR ' '

-* ** LINKED SEGMENTS FOR STATIC JOIN
  CLASSNAME STLINK PART_OF JOINST KEYS S0
    FIELDNAME LINK          ALIAS FCLASSNAME  FORMAT A12

-* ** JOIN STATIC ALL
  CLASSNAME JOINSTA PART_OF FLDSPEC KEYS S0
    FIELDNAME JOINFILE      ALIAS FQUALNAME  FORMAT A22    ACTUAL A22
    FIELDNAME JOINCLASS     ALIAS FCLASSNAME  FORMAT A22    ACTUAL A22
    FIELDNAME JOINFIELD     ALIAS FFIELDNAME  FORMAT A22    ACTUAL A22
    DEFINE RJOINALL/A66 = JOINFILE | JOINCLASS | JOINFIELD;
    FIELDNAME STA_ONLY      ALIAS LINKONLY    FORMAT A4      ACTUAL A4
      ACCEPT ONLY OR ' '

-* ** LINKED SEGMENTS FOR JOIN STATIC ALL
  CLASSNAME STALINK PART_OF JOINSTA KEYS S0
    FIELDNAME LINK          ALIAS FCLASSNAME  FORMAT A12

-* ** DYNAMIC JOIN
  CLASSNAME JOINDY PART_OF FLDSPEC KEYS S0
    FIELDNAME JOINFILE      ALIAS FQUALNAME  FORMAT A22    ACTUAL A22
    FIELDNAME JOINCLASS     ALIAS FCLASSNAME  FORMAT A22    ACTUAL A22
    FIELDNAME JOINFIELD     ALIAS FFIELDNAME  FORMAT A22    ACTUAL A22
    DEFINE JOIN/A66 = JOINFILE | JOINCLASS | JOINFIELD;
    FIELDNAME DY_ONLY       ALIAS LINKONLY    FORMAT A4      ACTUAL A4
      ACCEPT ONLY OR ' '

-* ** LINKED SEGMENTS FOR DYNAMIC JOINS
  CLASSNAME DYLINK PART_OF JOINDY KEYS S0
    FIELDNAME LINK          ALIAS FCLASSNAME  FORMAT A12

-* ** JOIN DYNAMIC ALL
  CLASSNAME JOINDYA PART_OF FLDSPEC KEYS S0
    FIELDNAME JOINFILE      ALIAS FQUALNAME  FORMAT A22    ACTUAL A22
    FIELDNAME JOINCLASS     ALIAS FCLASSNAME  FORMAT A22    ACTUAL A22
    FIELDNAME JOINFIELD     ALIAS FFIELDNAME  FORMAT A22    ACTUAL A22
    DEFINE JOINALL/A66 = JOINFILE | JOINCLASS | JOINFIELD;
    FIELDNAME DYA_ONLY      ALIAS LINKONLY    FORMAT A4      ACTUAL A4
      ACCEPT ONLY OR ' '

-* ** LINKED SEGMENTS FOR JOIN DYNAMIC ALL
  CLASSNAME DYALINK PART_OF JOINDYA KEYS S0
    FIELDNAME LINK          ALIAS FCLASSNAME  FORMAT A12

```

Querying the Fusion Catalog Master File

You can get detailed information about data source names and structures by issuing a query against the FUSCAT Master File in a stored procedure.

Example

Issuing a Query

The following query returns the names of all Master Files:

```
TABLE FILE FUSCAT
PRINT FILENAME
END
```

The next query shows how to return the names of all Master Files that satisfy a given set of conditions, where CND represents a valid logical expression:

```
TABLE FILE FUSCAT
PRINT FILENAME
WHERE (OR IF) CND
END
```

The following query lists all Master Files that contain the attribute FILETYPE FUSION:

```
TABLE FILE FUSCAT
PRINT FILENAME
WHERE FILETYPE EQ 'FUSION'
END
```

This query illustrates how to return the names of all Master Files, classes, and field names involved in a join:

```
TABLE FILE FUSCAT
PRINT FILENAME AND FIELDNAME AND JOIN AND RJOIN
      AND JOINALL AND RJOINALL
END
```

This query returns the names of all fields that are part of any MD index:

```
TABLE FILE FUSCAT
PRINT MDI_LOCATION_FOR_FIELD BY FIELDNAME BY FILENAME
END
```

This query returns the horizontal partitions for all Fusion data sources:

```
TABLE FILE FUSCAT
PRINT HORIZONTAL_PARTITION_FILE
WHERE FILETYPE EQ FUSION
END
```

APPENDIX A

Sample Schemas

Topics:

- Master Files for the XMDI1 Multi-Dimensional Index
- The ORDERS Data Source

The Fusion Master Files and Fusion Access Files for the Fusion data sources used in the examples presented throughout this manual are included in this topic.

Master Files for the XMDI1 Multi-Dimensional Index

Chapter 4, *Database Administration Facilities*, includes an example illustrating the creation of an MDI named XMDI1. This MDI has four data sources participating in it: ORDERSI1, CUSTOMER, PARTS, and SUPPLIER. The Master Files of these data sources are described here:

Fusion Master File ORDERSI1

```

FILENAME ORDERSI1  FILETYPE FUSION
CLASS  S_ORDERS  KEYS S1
      FIELD O_ORDERKEY      FORMAT I9      INDEX ON
      FIELD O_CUSTKEY      FORMAT I9      INDEX ON
                                           JOIN_TO CUSTOMER.C_CUST.C_CUSTKEY

      FIELD O_ORDERSTATU    FORMAT A1
      FIELD O_TOTALPRICE    FORMAT D15.2
      FIELD O_ORDERDATE     FORMAT YYMD      INDEX MD_EXTERNAL
                                           LOCATION XMDI1

      FIELD O_ORDERPRIOR    FORMAT A15
      FIELD O_CLERK         FORMAT A15
      FIELD O_SHIPPRIORI    FORMAT D10
      FIELD O_COMMENT       FORMAT A49

CLASS  L_LINE  PART_OF S_ORDERS  KEYS S1  TARGET_OF XMDI1
      FIELD L_LINENUMBER    FORMAT I9
      FIELD L_PARTKEY      FORMAT I9      INDEX ON
                                           JOIN_TO PARTS.P_PART.P_PARTKEY

      FIELD L_SUPPKEY      FORMAT I9
                                           JOIN_TO SUPPLIER.S_SUPP.S_SUPPKEY

      FIELD L_QUANTITY     FORMAT D10      INDEX MD_EXTERNAL
                                           LOCATION XMDI1

      FIELD L_EXTENDEDPR    FORMAT D15.2
      FIELD L_DISCOUNT    FORMAT D10
      FIELD L_TAX          FORMAT D10
      FIELD L_RETURNFLAG   FORMAT A1
      FIELD L_LINESTATUS   FORMAT A1
      FIELD L_SHIPDATE     FORMAT YYMD
      FIELD L_COMMITDATE   FORMAT YYMD
      FIELD L_RECEIPTDAT    FORMAT YYMD
      FIELD L_SHIPINSTRU   FORMAT A25
      FIELD L_SHIPMODE     FORMAT A10
      FIELD L_COMMENT      FORMAT A59

```

Fusion Master File CUSTOMER

```

FILENAME CUSTOMER FILETYPE FUSION
CLASS C_CUST KEYS S1
  FIELD C_CUSTKEY          FORMAT I9      INDEX ON
  FIELD C_NAME             FORMAT A25
  FIELD C_ADDRESS          FORMAT A25
  FIELD C_NATION           FORMAT A25      INDEX ON
  FIELD C_REGION           FORMAT A25      INDEX ON
  FIELD C_PHONE            FORMAT A15
  FIELD C_ACCTBAL          FORMAT D15.2
  FIELD C_MKTSEGMENT       FORMAT A10
                                     INDEX MD_EXTERNAL
                                     LOCATION XMDI1
  FIELD C_COMMENT          FORMAT A27
    
```

Fusion Master File SUPPLIER

```

FILENAME SUPPLIER FILETYPE FUSION
CLASS S_SUPP KEYS S1
  FIELD S_SUPPKEY ALIAS SUPPKEY  FORMAT I9      INDEX ON
  FIELD S_NAME             FORMAT A25
  FIELD S_ADDRESS          FORMAT A25
  FIELD S_NATION           FORMAT A25      INDEX ON
  FIELD S_REGION           FORMAT A25      INDEX MD_EXTERNAL
                                     LOCATION XMDI1
  FIELD S_PHONE            FORMAT A15
  FIELD S_ACCTBAL          FORMAT D15.2
  FIELD S_COMMENT          FORMAT A17
CLASS PS_PART PART_OF S_SUPP KEYS S1
  FIELD PS_PARTKEY ALIAS PARTKEY  FORMAT I9      INDEX ON
  FIELD PS_AVAILQTY        FORMAT D10
  FIELD PS_SUPPLYCOS        FORMAT D15.2  INDEX MD_EXTERNAL
                                     LOCATION XMDI1
  FIELD PS_COMMENT          FORMAT A124
    
```

Fusion Master File PARTS

```

FILENAME PARTS FILETYPE FUSION
CLASS P_PART KEYS S1
  FIELD P_PARTKEY ALIAS PARTKEY  FORMAT I9      INDEX ON
  FIELD P_NAME             FORMAT A50
  FIELD P_MFGR             FORMAT A25
  FIELD P_BRAND            FORMAT A10
  INDEX ON
  FIELD P_TYPE             FORMAT A25      INDEX MD_EXTERNAL
                                     LOCATION XMDI1
  FIELD P_SIZE             FORMAT D10      INDEX ON
  FIELD P_CONTAINER        FORMAT A10
  FIELD P_RETAILPRIC       FORMAT D15.2
  FIELD P_COMMENT          FORMAT A14
    
```

The ORDERS Data Source

In Chapter 1, *Introducing Fusion*, the ORDERS data source is used to illustrate a Fusion Schema structure. The data source is horizontally partitioned and includes predicates in its Fusion Access File. The ORDERS data source and its Access File are described here.

Fusion Master File ORDERS

```

FILE ORDERS FILETYPE FUSION ACCESSFILE ORDERS
CLASS S_ORDERS KEY S1
  FIELD ORDERKEY          FORMAT I9    INDEX ON
  FIELD CUSTKEY ALIAS CUSTKEY          FORMAT I9    INDEX ON
  RJOIN_TO CUSTOMER.CUSTKEY AS C
  FIELD ORDERSTATU        FORMAT A1
  FIELD TOTALPRICE        FORMAT D15.2
  FIELD ORDERDATE         FORMAT YYMD
  FIELD SHIPPRIORI        FORMAT D10
SECTION COMSEG
  FIELD COMMENT           FORMAT A49

CLASS L_LINE PART_OF S_ORDERS          KEYS S1    TARGET_OF MDIORDS
  FIELD L_LINENUMBER      FORMAT I9
  FIELD L_PARTKEY ALIAS PARTKEY          FORMAT I9
  INDEX MD_EXTERNAL
  LOCATION MDIORDS
  RJOIN_TO PARTS.PARTKEY AS P
  FIELD L_SUPPKEY ALIAS SUPPKEY          FORMAT I9
  RJOIN_TO SUPPLIER.S_SUPP.SUPPKEY LINK PS_PART AS S
  FIELD L_QUANTITY        FORMAT D10
  INDEX MD_EXTERNAL LOCATION MDIORDS
  FIELD L_DISCOUNT       FORMAT D10
  INDEX MD_EXTERNAL LOCATION MDIORDS
  FIELD L_TAX              FORMAT D10
  INDEX MD_EXTERNAL LOCATION MDIORDS
  FIELD L_LINESTATUS      FORMAT A1
  FIELD L_SHIPDATE        FORMAT YYMD
  INDEX MD_EXTERNAL LOCATION MDIORDS
SECTION SHIPINFO
  FIELD L_COMMITDATE      FORMAT YYMD
  FIELD L_RECEIPTDAT      FORMAT YYMD
  FIELD L_SHIPINSTRU      FORMAT A25
  FIELD L_SHIPMODE        FORMAT A10
  FIELD L_COMMENT         FORMAT A59

```

Fusion Access File ORDERS (UNIX Version)

MASTER ORDERS

```
DATANAME /home1/orders1.fus
  where ORDERDATE FROM '1995/01/01' TO '1995/01/31';
DATANAME /home1/orders2.fus
  where ORDERDATE FROM '1995/02/01' TO '1995/02/28';
DATANAME /home1/orders3.fus
  where ORDERDATE FROM '1995/03/01' TO '1995/03/31';
DATANAME /home1/orders4.fus
  where ORDERDATE FROM '1995/04/01' TO '1995/04/30';
DATANAME /home1/orders5.fus
  where ORDERDATE FROM '1995/05/01' TO '1995/05/31';
```

MDILOC MDIORDS

```
DATANAME /home1/mdiords.mdi
```

MASTER CUSTOMER

```
DATANAME /home1/cust.fus
```

MASTER PARTS

```
DATANAME /home1/parts.fus
```

MASTER SUPPLIER

```
DATANAME /home1/supplier.fus
```

APPENDIX B

Fusion Functions and Subroutines

Topics:

- Calling a Function or Subroutine
- Numeric Functions and Subroutines
- Date Functions and Subroutines
- Alphanumeric Subroutines

A function or subroutine is a set of calculations, stored under its own name, that returns a single value as a result of operating on values (called arguments) that you pass to it. You invoke a function by using its name and arguments in an expression. This topic lists the functions and subroutines available in Fusion.

For complete information about Fusion functions and subroutines, see *Using Functions and Subroutines* in *Creating Reports*. For more information about expressions, see Chapter 5, *Creating Derived Fields*.

Calling a Function or Subroutine

Functions and subroutines are accessed and invoked differently. Both return a single value as a result of the operation.

Syntax

How to Call a Function

```
function (arg1, arg2, ...)
```

where:

```
function
```

Is the name of the function.

```
arg1, arg2, ...
```

Are the arguments.

How to Call a Subroutine

```
subroutine (arg1, arg2, ... {outputfield|'format'})
```

where:

```
subroutine
```

Is the name of the subroutine.

```
arg1, arg2, ...
```

Are the arguments.

```
{outputfield|'format'}
```

Is the name of the output field or its format.

Numeric Functions and Subroutines

Numeric functions and subroutines enable you to perform numeric calculations on numeric constants and fields.

Performing Basic Numeric Calculations

The following table lists basic numeric functions that can be performed in Fusion.

Function	Description
ABS	Returns the absolute value of an argument.
EXPN	Evaluates an argument expressed in scientific notation.
INT	Returns the integer part of an argument.
LOG	Returns the logarithm of its argument.
MAX	Returns the maximum value from a list of arguments.
MIN	Returns the minimum value from a list of arguments.
SQRT	Returns the square root of an argument.

Performing Complex Numeric Manipulations

The following subroutines perform complex numeric manipulations:

Subroutine	Description
CHKPCK	Examines the value of a packed field to verify that it is in packed format.
EXP	Raises the number “e” to a power you specify.
MOD	There are three MOD functions. They calculate the remainder from a division and return the result as follows: IMOD returns an integer, FMOD returns a single-precision floating-point number, and DMOD returns a double-precision floating-point number.
PCKOUT	Writes packed numbers between one and eight bytes to extract files.
PRDNOR	Generates reproducible sets of double-precision random numbers that are normally distributed with an arithmetic mean of 0 and a standard deviation of 1.

Subroutine	Description
PRDUNI	Generates reproducible sets of double-precision random numbers uniformly distributed between 0 and 1.
RDNORM	Generates double-precision random numbers that are normally distributed with an arithmetic mean of 0 and a standard deviation of 1.
RDUNIF	Generates double-precision random numbers that are uniformly distributed between 0 and 1.

Changing the Format of a Numeric Value

You can convert a number from one format to another with one of the following subroutines. The EDIT function can also perform these conversions.

Subroutine	Description
ATODBL	Converts a number in alphanumeric format to FOCUS double-precision format.
FTOA	Converts a number in a FOCUS numeric format to alphanumeric format with edit options, such as commas and dollar signs.
ITOPACK	Converts a number in binary format to packed format.
ITOZ	Converts a number in integer format to zoned decimal format.
UFMT	Converts an alphanumeric string to hexadecimal format.

Date Functions and Subroutines

Date functions and subroutines manipulate date values stored in integer, packed, or alphanumeric format.

Calculating the Difference Between Dates

The following date functions calculate the difference between two dates:

Function	Description
DMY	Calculates the difference between two dates in day-month-year order.
MDY	Calculates the difference between two dates in month-day-year order.
YMD	Calculates the difference between two dates in year-month-day order.

Performing Arithmetic Calculations on Dates

The following subroutines perform arithmetic calculations on dates stored in integer, packed, or alphanumeric format:

Subroutine	Description
AYM	Adds and subtracts months from dates in year-month.
AYMD	Adds and subtracts days from dates in year-month-day.
CHGDAT	Rearranges the year, month, and day portions of dates, and converts dates between long and short date formats. Long formats contain the year, month, and day; short formats contain one or two of these elements.
DA	The DA functions convert dates to the corresponding number of days elapsed since December 31, 1900. DADMY converts dates in day-month-year format. DADYM converts dates in day-year-month format. DAMDY converts dates in month-day-year format. DAMYD converts dates in month-year-day format. DAYDM converts dates in year-day-month format. DAYMD converts dates in year-month-day format.
DOWK	The DOWK functions determine the day of the week (Sunday through Saturday) for dates. DOWK returns the day as a 3-letter abbreviation. DOWKL returns the full name of the day.
DT	The DT functions convert numbers representing the days elapsed since December 31, 1900 to corresponding dates. DTDYMY converts numbers to day-month-year dates. DTDYM converts numbers to day-year-month dates. DTMDY converts numbers to month-day-year dates. DTMYD converts numbers to month-year-day dates. DTYDM converts numbers to year-day-month dates. DTYMD converts numbers to year-month-day dates.
GREGDT	Converts dates in Julian format to year-month-day format.
JULDAT	Converts dates from year-month-day format to Julian (year-day) format.
YM	Finds the number of months between two dates in year-month format.

Alphanumeric Subroutines

Alphanumeric subroutines allow you to manipulate alphanumeric fields or character strings. The following table lists subroutines that manipulate alphanumeric fields or character strings.

Subroutine	Description
ARGLEN	Measures the length of a character string within a field, excluding trailing blanks.
CHKFMT	Checks for incorrect character types by comparing each character in the input string to the corresponding character in a mask.
CTRAN	Converts one character in a string to another character.
CTRFLD	Centers a character string within a field, excluding trailing blanks.
GETTOK	Divides a character string at a character called the delimiter and returns a substring called the token.
LCWORD	Converts uppercase characters in an alphanumeric string to lowercase, word by word.
LJUST	Left-justifies a character string within a field. All leading blanks become trailing blanks.
LOCASE	Converts uppercase characters in an alphanumeric string to lowercase.
OVERLAY	Overlays a substring on another character string.
PARAG	Divides lines of text into smaller lines with delimiters.
POSIT	Finds the starting position of a substring within a larger string.
REVERSE	Inverts and prints the contents of a field.
RJUST	Right-justifies a character string within a field. All trailing blanks become leading blanks.
SOUNDEX	Searches for a character string phonetically rather than by the way it is spelled.
SUBSTR	Extracts substrings, based on where they start and end in the parent string.
UPCASE	Converts lowercase characters in an alphanumeric string to uppercase.

In addition, you can convert a number to alphanumeric format with the EDIT or FTOA functions.

APPENDIX C

Using Data Types and Display Options

Topics:

- Numeric Format
- Alphanumeric Format
- Date Format

Fusion supports a wide range of formats. The USAGE attribute in the Fusion Master File identifies each field's data type; it can also describe how to display the field in a report. This topic explains Fusion data types and describes the display options available for each data type. For information about the Fusion Master File, see Chapter 2, *Understanding Your Fusion Schema*.

Numeric Format

Use a numeric format for a value to be interpreted as a number. There are four types of numeric formats: integer, double-precision floating-point, single-precision floating-point, or packed decimal.

Integer Format

Use integer format to display whole numbers—that is, any value composed of the digits zero to nine, without a decimal point. The format is

`In`

where:

`I`

Indicates integer format.

`n`

Is the maximum number of digits, up to eleven.

Examples

Displaying a Number in Integer Format

The following contains examples of numbers displayed in integer format:

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

Double-Precision Floating-Point

Use double-precision floating-point format to display a value composed of the digits zero to nine and an optional decimal point. The format is

`Dn[.s]`

where:

`D`

Indicates double-precision floating-point format.

`n`

Is the maximum number of characters to display. This value includes digits, a leading minus sign if the field may contain negative values, and a decimal point if one is to display. The maximum value permitted is 17.

`s`

Is the number of digits that are decimal places. This value is optional.

Examples

Displaying a Number in Double-Precision Floating-Point Format

The following are examples of numbers in double-precision floating-point format:

Format	Display
D8.2	3,187.54
D8	416

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

Single-Precision Floating-Point

Use single-precision floating-point format for any value composed of the digits zero to nine, including an optional decimal point. This format is intended for use with smaller decimal numbers; unlike double-precision floating-point format, its length cannot exceed nine positions. The format is

`Fn[.s]`

where:

F

Indicates single-precision floating-point format.

n

Is the maximum number of characters to display. This value includes digits, a leading minus sign if the field may contain negative values, and a decimal point if one is to display. The maximum value permitted is 9.

s

Is the number of digits that are decimal places. This value is optional.

Examples

Displaying a Number in Single-Precision Floating-Point Format

The following are examples of numbers in single-precision floating-point format:

Format	Display
<code>F5.1</code>	<code>614.2</code>
<code>F4</code>	<code>318</code>

Packed Decimal

Use packed decimal format for any value composed of the digits zero to nine, including an optional decimal point. The format is

`Pn[.s]`

where:

`P`

Indicates packed decimal format.

`n`

Is the maximum number of characters to display. This value includes digits, a leading minus sign if the field may contain negative values, and a decimal point if one is to display. The maximum number of characters allowed is 33.

`s`

Is the number of digits that are in decimal places. This value is optional.

Using a Packed Decimal Format

The following are examples of numbers in packed decimal format:

Format	Display
<code>P9.3</code>	<code>4168.368</code>
<code>P7</code>	<code>617542</code>

Numeric Display Options

Use display options to edit numeric formats. Display options affect how the data in the field is printed or appears on the screen, but not how the data is stored in your file.

The following table explains the display options and provides examples:

Option	Format	Description
Comma inclusion	<i>InC</i>	Inserts a comma before every third significant digit. For example <i>41376</i> displays as: <i>41,376</i>
Zero suppression	<i>DnS</i>	Displays a value of 0 as a blank.
Bracket negatives	<i>InB</i>	Encloses a negative value in parentheses instead of including a leading negative sign. For example <i>-64187</i> displays as: <i>(64187)</i>
Credit negative	<i>InR</i>	Displays the letters CR after a negative value, instead of including a leading negative sign. For example <i>-3167</i> displays as: <i>3167 CR</i>
Leading zeros	<i>FnL</i>	For a number that contains fewer digits than the field's format designates as a maximum, leading zeros fill in any available spaces to meet the maximum. For example, with the format F4L, <i>31</i> displays as: <i>0031</i>

Option	Format	Description
Floating dollar	<code>IrM</code>	Includes a dollar sign to the left of the first significant digit, and adds a comma before every third significant digit. For example <code>6148</code> displays as: <code>\$6,148</code>
Non-floating dollar	<code>IrN</code>	Includes a dollar sign to the left of the field, and adds a comma before every third significant digit. For example <code>5432</code> with a field length of 7 would display as: <code>\$ 5,432</code>
Scientific notation	<code>Dn.sE</code>	Displays a number in scientific notation. For example <code>1234.5</code> displays as: <code>0.123456D+04</code>

Combining Display Options

You can combine several display options. You can specify the options 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).

Example

Combining Display Options

In this example, the comma inclusion and bracket negatives options are combined. The combined format is `I5CB`. Therefore, the value

`-61874`

displays as:

`(61,874)`

Rounding Values

If a numeric field is assigned a value with more decimal places than are indicated in the field's format, Fusion rounds the value to an acceptable size. Fusion rounds down when the first extra decimal digit is less than five, and rounds up when it is five or greater.

Fusion handles the details of rounding in the following way for each numeric format:

- **Integer format.** When a value containing decimal places is assigned to an integer field through a MODIFY transaction, the value is rounded before it is stored. If the value is assigned through a derived field definition, the decimal portion of the value is truncated before it is stored.
- **Packed decimal format.** The value is rounded before it is stored.
- **Double- and single-precision floating-point formats.** The full value is stored in the field up to the limit of precision determined by the field's internal storage type. The value is rounded before it is displayed.

If the hexadecimal representation of a floating-point value is a non-terminating decimal, it is stored as a non-repeating slightly lower number. If the value to be rounded is a five, it will become a four.

Examples

Rounding a Decimal

Suppose you assign the value 746.1289 to a packed-decimal field defined with two decimal places:

```
FIELDNAME PRESSURE FORMAT P8.2
```

Since the first extra digit—that is, the first one past the specified length of two decimal places—is 8, and since 8 is greater than or equal to five, the value is rounded up to 746.13.

Rounding a Non-Terminating Decimal

Consider a floating-point double-precision field with one decimal place:

```
FIELD VELOCITY FORMAT D5.1
```

If you assign the value 1.15 to VELOCITY, the value is stored as 1.149999 due to the nature of floating-point arithmetic. While the original number, 1.15, would have been rounded up to 1.2 (since the first extra digit was 5 or greater), the number actually stored is rounded down to 1.1.

Alphanumeric Format

Use the alphanumeric format for any value to be interpreted as a sequence of characters and composed of any combination of digits, letters, and other characters. The format is

`An`

where:

`A`

Indicates alphanumeric format.

`n`

Is the maximum number of characters in the field. The maximum value permitted is 256.

Examples

Using Alphanumeric Format

The following are examples of values displayed in alphanumeric format:

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

Changing the Display of an Alphanumeric Value

The standard numeric display options are not available for the alphanumeric data format. However, you can use the EDIT function to supply a pattern for alphanumeric data.

Using EDIT to Change the Display

For instance, to display the value of the field PCODE with each part separated by a hyphen, use the following expression to create a derived field with the desired format:

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

Date Format

Use the date format to define a field with the date components you need, and you can manipulate and display the field's value in ways appropriate to a date. Date formats enable you to:

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

- Do arithmetic with dates and compare dates without resorting to special date-handling functions.
- Refer to dates in a natural way, such as JAN 1 1995, without regard to display or editing formats.
- Automatically validate dates in transactions.

Date Display Options

Date format doesn't include a data type or length; instead, it specifies date component options (D, W, M, Q, Y, and YY) and display options. These are explained in the following table:

Option	Value/Function	Description
D	Day	Prints a value from 1 to 31 for the day.
M	Month	Prints a value from 1 through 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 the three-letter abbreviation for a month or day in uppercase, if M or D is included in the FORMAT specification.
t	Translate month or day	Prints the three-letter abbreviation for a month or day, capitalizing the first letter only.*
TR	Translate month or day	Prints the full name of a month or day in uppercase.
tr	Translate month or day	Prints the full name of a month or day, capitalizing the first letter only.*
Q	Quarter	Prints the quarter. Prints only a number if Q is specified by itself. Prints Q with the number if it is specified with another date format option.
W	Day-of-Week	A three-letter abbreviation of the day of the week is printed in uppercase if it is included in a FORMAT specification with other date component options. The number of the day of the week (1-7, Mon=1) is printed if it is the only date component

Option	Value/Function	Description
		option in the FORMAT specification.
w	Day-of-Week	A three-letter abbreviation of the day of the week is printed, capitalizing the first letter, if it is included in a FORMAT specification with other date component options. The number of the day of the week (1-7, Mon=1) is printed if it is the only date component option in the FORMAT specification.
WR	Day-of-Week	The day of the week is printed in uppercase if it is included in a FORMAT specification with other date component options. The number of the day of the week (1-7, Mon=1) is printed if it is the only date component option in the FORMAT specification.*
wr	Day-of-Week	The day of the week is printed, capitalizing the first letter, if it is included in a FORMAT specification with other date component options. The number of the day of the week (1-7, Mon=1) is printed if it is the only date component option in the FORMAT specification.*
JUL	Julian format	Prints the date in Julian format.

***Note:** When using these display options, ensure they are stored as lowercase letters in the file description.

Examples

Using Date Display Options

The following table illustrates the use of several options with the value 1 (January) for the month.

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

Displaying Separating Characters

Control the date separators displayed with the date by including different separators in the format specification. In YMD, MDYY, and YM formats, the date is displayed with a slash separating them. In YQ and QYY format, the date is displayed with the year and quarter separated by a blank. Single component formats display the single number or name.

You can change the separating character to a period, a dash, a blank, or you can eliminate it entirely. Placing a symbol for the separator where you would like it to display does this. The options are:

- . Uses a period as the separating character.
- Uses a hyphen as the separating character.
- B Uses a blank space as the separating character.
- | Uses a concatenation symbol as the separating character.

Example

Including a Separating Character

The following table includes examples based on a YMD format that displays the number with the designated separating character.

Usage	Display
YMD	93/12/24
Y.M.D	93.12.24
Y-M	93-12
YBMBD	93 12 24
Y M D	93 12 24

Natural Date Literals and Numeric Date Literals

You can enter a date as a natural date literal or as a numeric date literal. Fusion automatically validates any field formatted as a date before entering it in the data source.

Natural Date Literals

A natural date literal enables you to specify a date in a natural, easily understandable way. This is done by including spaces between date components and using abbreviations for month names instead of the number representing the month. Natural date literals can be used in all date computations.

There are several available formats for natural date literals. The following table explains these formats:

Format	Description
Year-month-day	Displays the four-digit year, uppercase three-character abbreviation or full name of the month, and a one- or two-digit day of the month.
Year-month	Displays the four-digit year, and the three-character abbreviation or full name of the month, in uppercase.
Year-quarter	Displays the four-digit year, and the letter Q plus the quarter number.
Month	Displays the three-character abbreviation or full name of the month, in uppercase.
Quarter	Displays a Q and the quarter number.
Day of week	Displays the three-character abbreviation or full name of the day, in uppercase.

You can specify the date components of a natural date literal in any order, regardless of their order in the USAGE format of the target field. Separate date components with one or more blanks. For example, if the USAGE format is YM, MAY 1993 and 1960 APR are both valid literals.

See Chapter 5, *Creating Derived Fields*, for more information on natural date literals.

Example

Using Natural Date Literals

With the natural date literals format, you can use any of the following to represent April 25, 1999:

```
APR 25 1999
```

```
25 APR 1999
```

```
1999 APR 25
```

Numeric Date Literals

A numeric date literal is a string 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, April 25 1960 must include 25 for the date, 04 for the month, and 60 for the year, in that order. This would be written as 250460.

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

For example, the difference between two date fields, NEWDATE and OLDDATE, can be found by subtracting one field from the other (NEWDATE - OLDDATE). If NEWDATE is defined as MtrD, OLDDATE must also be defined with date components M (month) and D (day); it cannot include Y (year) or Q (quarter), or exclude M or D.

Arithmetic expressions that assign values to quarters, months, or days of the week are computed modulo 4, 12, and 7, respectively, to avoid anomalies like fifth quarters and thirteenth months.

Examples

Manipulating a Numeric Date Literal

If THISQUARTER has the value 2 and a USAGE format of Q, the following field definition assigns NEWQUARTER a value of 1 (that is, the remainder of 5 divided by 4):

```
DEFINE NEWQUARTER/Q = THISQUARTER + 3;
```

Converting Date Fields

You can convert date fields. Two types of conversion are possible:

- **Format conversion.** Assigns the value of a date format field to an alphanumeric or integer field that uses date display options; the reverse conversion is also possible.
- **Date component conversion.** Assigns a field whose USAGE format includes one set of date components to another field with different components.

Converting a Date Field

If you assign the value of REPORTDATE (DMY) to ORDERDATE (Y), the year is extracted from REPORTDATE. If REPORTDATE is Apr 27 89, ORDERDATE is 89.

You can also assign the value of ORDERDATE to REPORTDATE; if the value of ORDERDATE is 89, the value of REPORTDATE would be Jan 1 89. In this case, the original date is rounded up to give a value to the missing date components.

How Date Fields Are Represented in Fusion

Fusion stores date fields as 4-byte binary integers that contain the elapsed time since the base date, December 31, 1900. For each field, the unit of elapsed time is that field's smallest date component.

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.

The date format's internal representation does not affect the end user, except to note that all dates set to the base date display as blanks, and all date fields which are entered as blank or as all zeros are accepted during validation and interpreted as the base date and displayed as blanks.

Example

Representing a Date Internally

For example, if REPORTDATE is defined as MDY, then elapsed time is measured in days, and internally the field contains the number of days between the entered date and the base date. If you enter the numeric literal for February 13, 1964 (that is, 021364), and then print the field in a report, 02/13/64 is displayed. Consider the following example:

```
DEFINE DAYS/I3 = 'FEB 13 1964' - REPORTDATE;
```

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

Date Format Support

Date format fields can be used with the following facilities:

- **Dialogue Manager.** An amper variable can function as a date field if it is formatted as a natural date literal. For example:

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

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

- **Extract files.** A date field in a SAVB and unformatted a HOLD file is stored as a 4-byte binary integer representing the time difference between the field's display value and the standard Fusion base date. Date fields in a SAVE file and formatted a HOLD file (for example, FORMAT WP) are stored as ASCII fields representing the field's face value, without any display options.

Representing a Date as an Alphanumeric, Integer, or Packed Decimal

You can represent a date as an alphanumeric, integer, or packed decimal field with date display options (D, M, Y, and T). Note, however, that this technique does not offer the full date support that is provided by the standard date format.

Alphanumeric and integer fields used with date display options acquire some date functionality when used with special date functions, as described in Appendix B, *Fusion Functions and Subroutines*.

When you define a date as an alphanumeric or integer field with date display options, you can specify the year, month, and day. If all three of these elements are present, the date has six digits (eight if the year is represented as four digits):

Format	Display
I6MDY	04/21/76
A6YMD	76/04/21
P6DMY	21/04/76
I8DMYY	21/04/1976

You can translate a month number (1 to 12) to the corresponding month name by adding the letter T to the format, immediately after the M. For example:

Format	Data	Display
I6MTDY	05/21/76	MAY 21 76
I4MTY	0676	JUN 76
I2MT	07	JUL

If a date only has the month element, a format of I2MT displays the value 4 as APR. In reports whose columns or rows are sorted by month, they appear in correct calendar order: JAN, FEB, MAR, because the sorting is based on the numerical, not alphabetical, values. (Note that without the T display option, I2M is interpreted as an integer with a floating dollar sign.)

Index

Symbols

- ? FDT command, 4-20
- ? FILE command, 4-20
- ? MDI command, 3-4
- ? PASS command, 6-5

A

- ABS function, B-3
- ACCEPTVALUES attribute, 2-24
- ACCESS attribute, 6-7, 6-10
- ACCESSFILE attribute, 2-7, 4-3
- ACTUAL attribute, 2-21
- ALIAS attribute, 2-20
- alphanumeric expressions, 5-4, 5-11
- alphanumeric format, C-8
- alphanumeric formats
 - display options, C-8
- alphanumeric subroutines, B-6
 - LCWORD, B-6
- ARGLEN subroutine, B-6
- arithmetic operators, 5-6
- ATODBL subroutine, B-4
- attributes, 2-2
 - class, 2-9
 - field, 2-16
 - file, 2-5, 2-34
- AUTOINDEX command, 3-6
- AYM subroutine, B-5
- AYMD subroutine, B-5

B

- Boolean expressions, 5-14

C

- character strings, 5-11
- CHGDAT subroutine, B-5
- CHKFMT subroutine, B-6
- CHKPCK subroutine, B-3
- class attributes, 2-9
 - CLASSKEYS, 2-11
 - CLASSNAME, 2-11
 - LOCATION, 2-13
 - PART_OF, 2-14
 - SECTION, 2-15
 - SUBCLASS_OF, 2-14
 - TARGET_OF, 2-13
- class declaration, 2-9
- classes, 1-7
 - joining, 2-30
- CLASSKEYS attribute, 2-11
- CLASSNAME attribute, 2-11
- commands
 - ? FDT, 4-20
 - ? FILE, 4-20
 - ? MDI, 3-4
 - ? PASS, 6-5
 - AUTOINDEX, 3-6
 - ENCRYPT, 6-15
 - HOLD FORMAT FUSION, 4-7, 4-11
 - MDICARDWARN, 4-19
 - MDIENCODING, 4-19
 - MDIPROGRESS, 4-18
 - PASS, 6-16
 - REBUILD, 4-26
 - REBUILD MIGRATE, 4-6
 - SET PASS, 6-6

- SET USER, 6-6
- computer systems, 1-2
- conditional expressions, 5-4, 5-14
- CTRAN subroutine, B-6
- CTRFLD subroutine, B-6
- D**
- DA subroutine, B-5
- data sources, 2-2
 - Fusion, 1-7
 - migrating to Fusion, 4-3, 4-7
 - multi-dimensional, 1-4
 - partitioned, 1-10
- database administration, 4-2
 - tools, 1-11
- database administrator, 6-3
 - DBA attribute, 6-3
- database security, 6-2
 - central security file (DBAFIELD), 6-12
 - database administrator, 6-3
 - user access, 6-5
- DATANAME attribute, 2-37
- DATASET attribute, 2-7, 4-3
- date constant, 5-9
- date expressions, 5-4, 5-7
 - combining fields, 5-10
 - date constant, 5-9
 - date format, 5-9
 - extracting date components, 5-10
 - field formats, 5-7, 5-9
 - performing calculations, 5-8
- date fields, C-12
 - converting, C-14
 - Fusion, C-14
- date formats, 5-7, 5-9, C-9
 - display options, C-9
 - natural date literals, C-12
 - numeric date literals, C-12
 - support, C-15
- date functions, B-4
- date subroutines, B-4
- dates, C-16
 - as alphanumeric, C-16
 - as integer, C-16
 - as packed decimal, C-16
- DBA attribute, 6-3
- DBAFIELD, 6-12
- declarations, 2-2
 - class, 2-9
 - field, 2-16
 - file, 2-34
 - group, 2-29
 - types, 2-3
- DEFCENT attribute, 2-22
- derived fields, 1-8, 5-2
 - with missing values, 5-3
- DESCRIPTION attribute, 2-26
- dimensional join, 3-7
- display options, C-8
 - bracket negatives, C-5
 - combining, C-6
 - comma inclusion, C-5
 - credit negative, C-5
 - date, C-9
 - floating dollar, C-6
 - leading zeros, C-5
 - non-floating dollar, C-6
 - scientific notation, C-6
 - separating characters, C-11
 - zero suppression, C-5
- DMY function, B-4
- double-precision floating-point format, C-2
- DOWK subroutine, B-5
- DT subroutine, B-5

E

ENCRYPT command, 6-15

EXP subroutine, B-3

EXPN function, B-3

expressions, 5-4

 alphanumeric, 5-4, 5-11

 conditional, 5-4, 5-14

 date, 5-4, 5-7

 field formats, 5-5

 logical, 5-4, 5-12

 numeric, 5-4, 5-5

F

FDFCENT attribute, 2-8

field attributes, 2-16

 ACCEPTVALUES, 2-24

 ACTUAL, 2-21

 ALIAS, 2-20

 DFCENT, 2-22

 DESCRIPTION, 2-26

 FIELDNAME, 2-19

 GROUPNAME, 2-29

 HELPMESSAGE, 2-25

 INDEX, 2-26

 JOIN_TO, 2-30

 JOIN_TO_ALL, 2-30

 LINK_TO, 2-17, 2-30

 LOCATION, 2-26

 MAXVALUES, 2-23, 2-26

 MISSING, 2-24

 ONLY, 2-17

 RJOIN_TO, 2-17

 TITLE, 2-23

 USAGE, 2-20, 2-29

 WITH, 2-17

 YRTHRESH, 2-22

field declarations, 2-16

field formats, 5-7

 date, 5-7

FIELDNAME attribute, 2-19

file attributes, 2-5, 2-34

 ACCESSFILE, 2-7

 DATANAME, 2-37

 DATASET, 2-7

 FDFCENT, 2-8

 FILENAME, 2-6

 FILETYPE, 2-6

 FYRTHRESH, 2-8

 INDEXLOCATION, 2-41

 LOCATION, 2-38

 MASTERNAME, 2-37

 MDILOCATION, 2-41

 REMARKS, 2-8

 WHERE, 2-39

file declarations, 2-34

FILENAME attribute, 2-6

FILETYPE attribute, 2-6

FOCUS data sources, 4-7

 migrating to Fusion, 4-7

formats, C-1

 alphanumeric, C-8

 date, C-9

 numeric, C-2

FTOA subroutine, B-4

functions, B-2

 date, B-4

 numeric, B-3

FUSCAT Master File, 1-11, 7-2

 describing, 7-2

 querying, 7-6

Fusion, 1-6

 Access Files, 1-8, 2-2, 2-33

 data sources, 1-7

 date fields, C-14

 declarations, 2-2

 encoding values, 3-11

 features, 1-9

 Master Files, 2-2, 5-2

 multi-dimensional index, 1-9

 partitions, 1-10

 schema, 1-8, 2-2

security, 6-2

Fusion Access Files, 1-8, 2-2, 2-33

file attributes, 2-34

horizontal partitioning, 2-33

joins, 2-33, 2-42

partitions, 4-11

Fusion catalog Master File (FUSCAT), 7-2

Fusion data sources, 2-2

classes, 1-7

derived fields, 5-2

describing, 2-37

large, 4-26

ORDERS data source, A-4

partitioning, 3-13

populating, 4-2

querying, 4-20

Fusion Master Files, 2-2, 5-2

class attributes, 2-9

declarations, 2-3

ENCRYPT command, 6-15

file attributes, 2-5

FUSCAT Master File, 7-2

guidelines, 2-4

security section, 6-2

Fusion Schema, 1-8, 2-2

FYRTHRESH attribute, 2-8

G

GETTOK subroutine, B-6

GREGDT subroutine, B-5

group declarations, 2-29

GROUPNAME attribute, 2-29

H

HELPMESSAGE attribute, 2-25

HOLD FORMAT FUSION command, 4-7, 4-11, 4-12

horizontal partitions, 1-10, 2-39, 3-13

I

INDEX attribute, 2-26

indexes, 3-2

b-tree, 1-3, 2-26

creating, 3-2

dimensions, 3-2

multi-dimensional, 2-26

non-intrusive external index, 4-20

partitioning, 4-17

strategies, 3-2

INDEXLOCATION attribute, 2-41

INT function, B-3

integer format, C-2

ITOPACK subroutine, B-4

ITOZ subroutine, B-4

J

JOIN_TO attribute, 2-30

JOIN_TO_ALL attribute, 2-30

joins, 1-3, 1-8, 2-30, 3-7

choosing fields, 3-8

describing, 2-42

dimensional, 3-7

non-unique join, 2-30

unique join, 2-30

JULDAT subroutine, B-5

-
- L**
- LCWORD subroutine, B-6
 - LINK_TO attribute, 2-30
 - LJUST subroutine, B-6
 - LOCASE subroutine, B-6
 - LOCATION attribute, 2-13, 2-26, 2-38
 - LOG function, B-3
 - logical expressions, 5-4, 5-12
 - Boolean expressions, 5-14
 - relational expressions, 5-12
- M**
- Master Files, 1-8
 - migrating to Fusion, 4-6
 - MASTERNAME attribute, 2-37
 - MAX function, B-3
 - MAXVALUES attribute, 2-23, 2-26
 - MDICARDWARN command, 4-19
 - MDIENCODING command, 3-11, 4-19
 - MDILOCATION attribute, 2-41
 - MDIPROGRES command, 4-18
 - MDY function, B-4
 - metadata, 1-11
 - MIN function, B-3
 - MISSING attribute, 2-24, 5-3
 - MOD subroutine, B-3
 - multi-dimensional data sources, 1-4, 1-5
 - multi-dimensional indexes, 1-4, 1-8, 1-9, 2-41, 3-2, 3-3
 - ? MDI command, 3-4
 - AUTOINDEX command, 3-6
 - creating, 3-2
 - displaying messages, 4-19
 - encoding values, 3-11
 - guidelines, 3-3
 - joining, 3-7
 - joining to, 3-8
 - MDICARDWARN command, 4-19
 - MDIENCODING command, 4-19
 - MDIPROGRESS command, 4-18
 - partitioning, 4-17
 - querying, 3-4, 4-18
 - REBUILD command, 4-14
 - retrieving data, 4-19
 - SQL queries, 3-4
 - TABLE queries, 3-3
 - XMDI1, A-2
- N**
- natural date literals, C-12
 - non-intrusive external index, 4-20
 - numeric date literals, C-12, C-13
 - numeric expressions, 5-4, 5-5
 - arithmetic operators, 5-6
 - order of evaluation, 5-6
 - numeric formats
 - display options, C-5
 - double-precision-floating point format, C-2
 - integer format, C-2
 - packed decimal format, C-4
 - rounding values, C-7
 - single-precision floating-point format, C-3
 - numeric formats:, C-2
 - numeric functions, B-3
 - LOG, B-3
 - numeric subroutines, B-3

O

order of evaluation, 5-6
OVLAY subroutine, B-6

P

packed decimal format, C-4
PARAG subroutine, B-6
PART_OF attribute, 2-14
partitioned data sources, 3-13
partitions, 1-10, 4-11
 horizontal, 3-13
 multi-dimensional index, 4-17
 updating, 4-27
 vertical, 3-13, 3-15
-PASS command, 6-16
passwords, 6-15
 variable, 6-16
PCKOUT subroutine, B-3
POSIT subroutine, B-6
PRDNOR subroutine, B-3
PRDUNI subroutine, B-4
procedures, 1-11
 encrypted, 6-16
 security, 6-15

Q

queries, 3-3
 ? FDT, 4-20
 ? FILE, 4-20
 ? MDI, 3-4
 ? PASS, 6-5

R

RDNORM subroutine, B-4
RDUNIF subroutine, B-4
REBUILD command, 4-7, 4-14, 4-26
REBUILD MIGRATE command, 4-6
 environment variables, 4-8
relational expressions, 5-12
REMARKS attribute, 2-8
RESTRICT attribute, 6-8, 6-10
REVERSE subroutine, B-6
RJUST subroutine, B-6

S

schema, 2-2
SECTION attribute, 2-15
security, 6-1
 database, 6-2
 stored procedures, 6-15
SET PASS command, 6-6
SET USER command, 6-6
single-precision floating-point format, C-3
SmartLoad, 4-27
 updating partitions, 4-27
SOUNDEX subroutine, B-6
SQL queries, 3-4
SQRT function, B-3
stored procedure security, 6-15
SUBCLASS_OF attribute, 2-14
subroutines, B-2
 alphanumeric, B-6
 date, B-4
 numeric, B-3
SUBSTR subroutine, B-6

T

TABLE queries, 3-3
TARGET_OF attribute, 2-13
TITLE attribute, 2-23

U

UFMT subroutine, B-4
UPCASE subroutine, B-6
USAGE attribute, 2-20, 2-29
USER attribute, 6-5
users, 6-5

- ACCESS attribute, 6-7, 6-10
- identifying, 6-5, 6-6
- limiting access, 6-8, 6-10
- querying, 6-5
- RESTRICT attribute, 6-8, 6-10
- specifying access, 6-7

V

values

- encoding, 3-11
- rounding, C-7

vertical partitions, 1-10, 3-13, 3-15

W

WHERE attribute, 2-39
WITHIN modifier, 2-27

Y

YM subroutine, B-5
YMD function, B-4

YRTHRESH attribute, 2-22

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. Send comments to:

Corporate Publications
Attn: Manager of Documentation Services
Information Builders
Two Penn Plaza
New York, NY 10121-2898

or FAX this page to (212) 967-0460, or call **Cristin Keely** at (212) 736-4433, x**3689**.

Name: _____

Company: _____

Address: _____

Telephone: _____ Date: _____

Comments:

Reader Comments